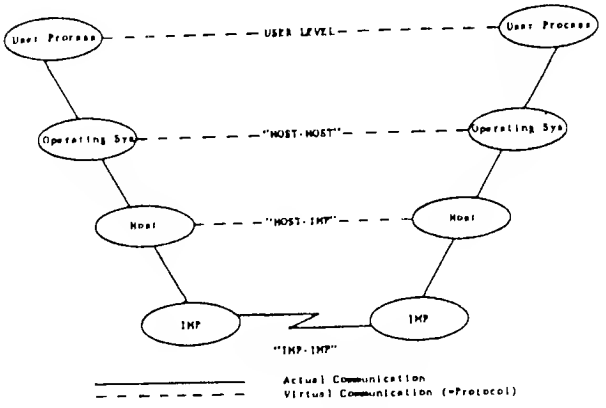


Figure 3—The layers of protocol

tions are often provided (e.g., saving a transcript of a session in a local file, sending a file in place of user-typed input, reporting whether various HOSTs are or have been up).

In the serving HOST it is desirable that a process controlled over the ARPANET behave as it would if controlled locally. The cleanest way to achieve this goal is to generalize the terminal control portion (TCP) of the operating system to accept ARPANET terminal interaction. It is unpleasant to modify any portion of a working computer system and modification could be avoided if it were possible to use a non-supervisor process (e.g., "server-TELNET" or "LOGGER") to perform the job creation, login, terminal input-output, interrupt, and logout functions in exactly the same way

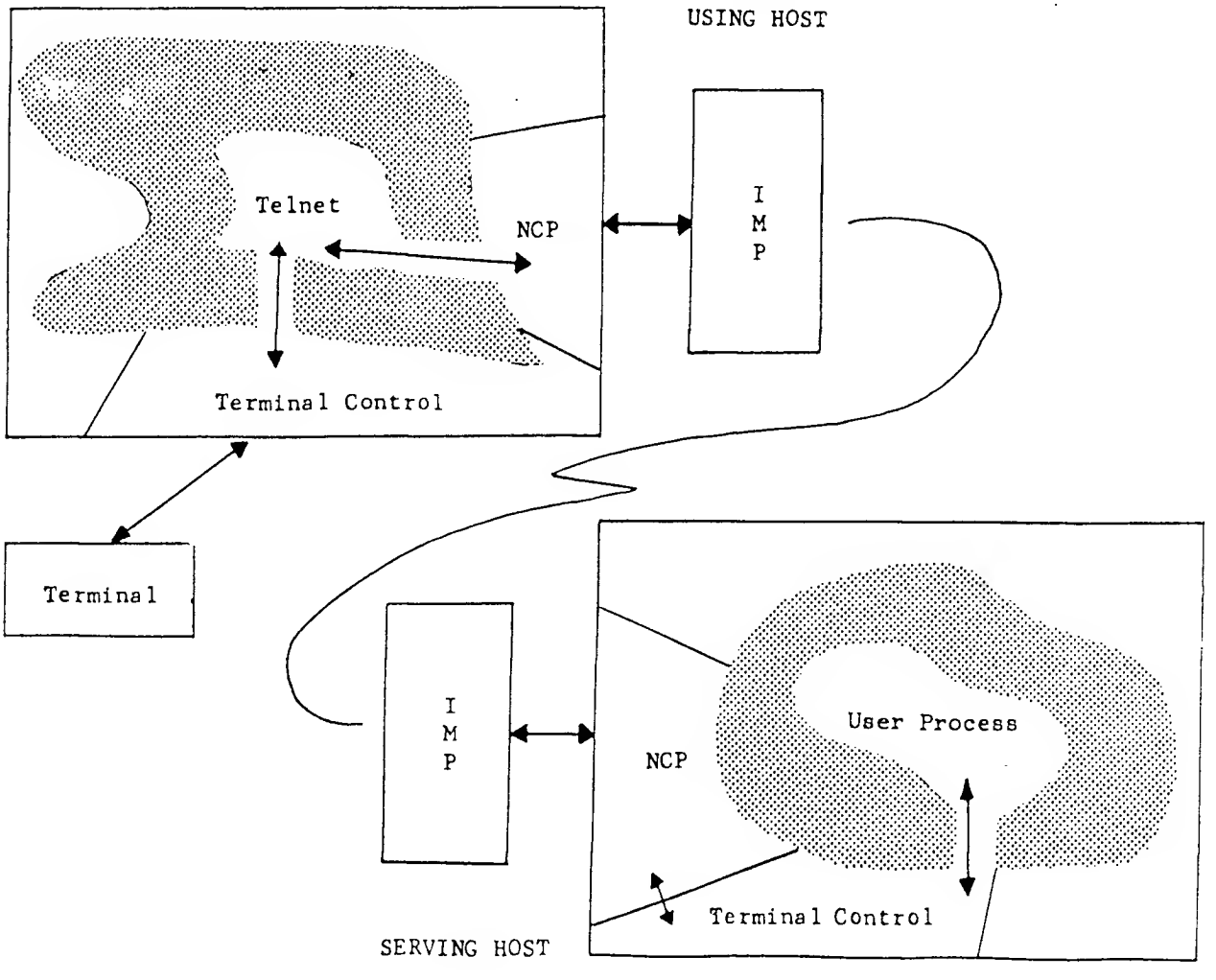


Figure 4—Data flow for remote interactive use

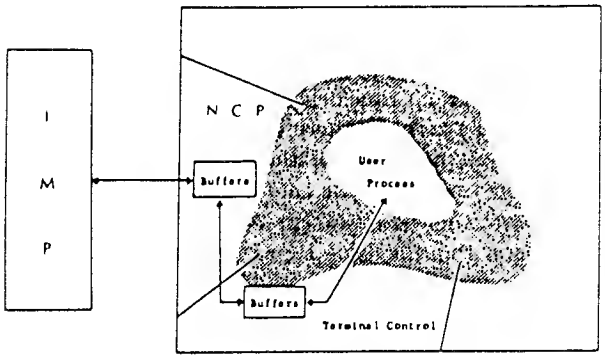


Figure 5—Data flow scheme for server

as a direct console user. Prior to the development of the ARPANET, no operating system provided these functions to non-supervisor processes in anywhere near the required completeness. Some systems have since been modified to support this generalized job control scheme. See Figures 5 and 6.

Efforts to standardize communications in the TELNET protocol focused on four issues: character set, echoing, establishing connections, and attention handling.

The chosen character set is 7-bit ASCII in 8-bit fields with the high-order bit off. Codes with the high-order bit on are reserved for TELNET control functions. Two such TELNET control function codes are the "long-space" which stands for the 200 millisecond space generated by the teletype BREAK button, and the synchronization character (SYNCH) discussed below in conjunction with the purpose of the TELNET interrupt signal.

Much controversy existed regarding echoing. The basic problem is that some systems expect to echo, while some terminals always echo locally. A set of conventions and signals was developed to control which side of a TELNET connection should echo. In practice, those systems which echo have been modified to include provision for locally echoing terminals. This is a non-trivial change affecting many parts of a serving HOST. For example, normally echoing server HOSTs do not echo passwords so as to help maintain their security. Terminals which echo locally defeat this strategy, how-

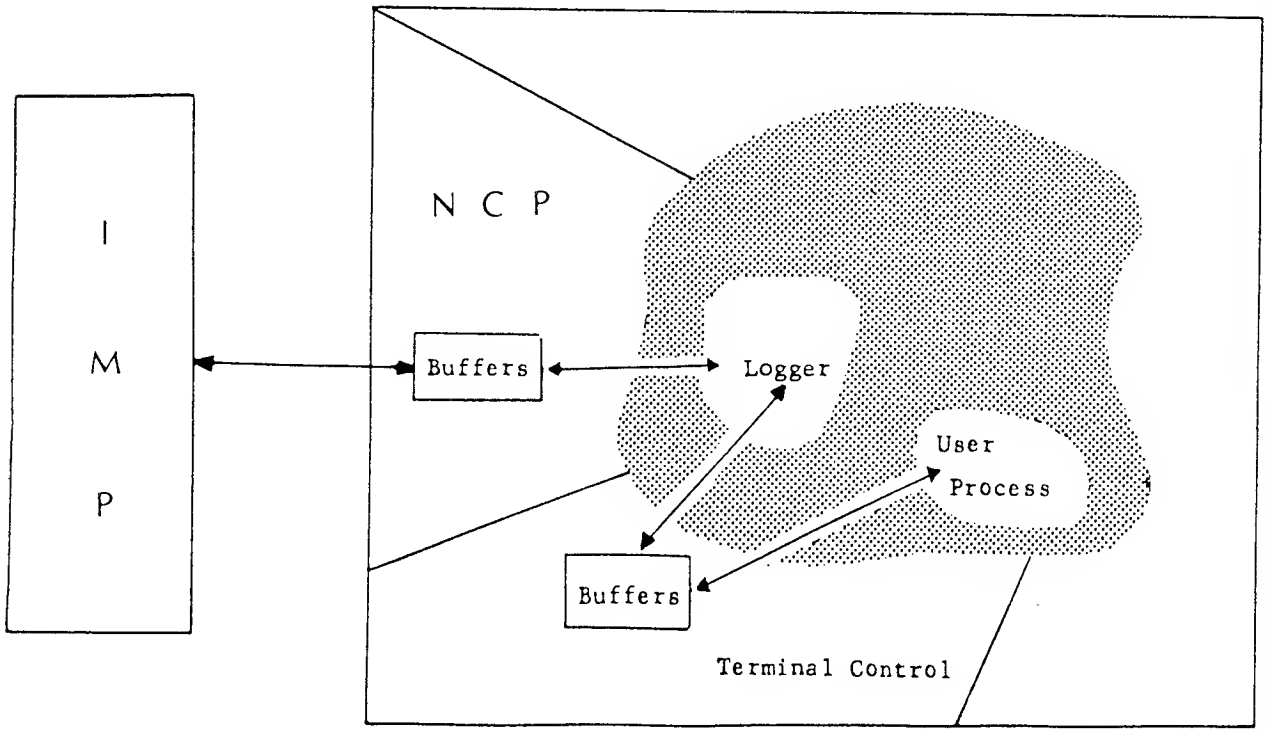


Figure 6—Alternate data flow scheme for a server

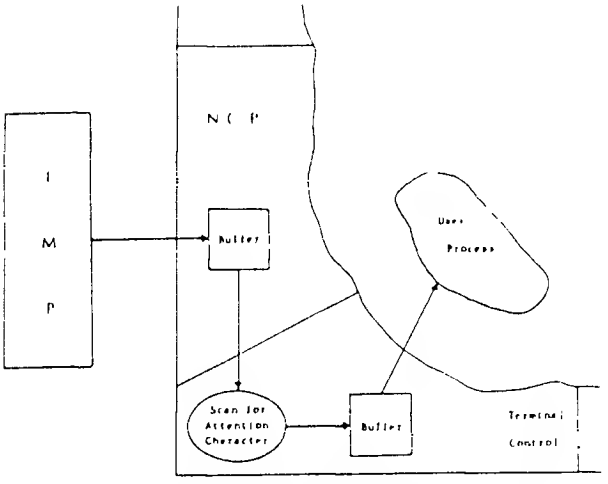


Figure 7—Data flow and processing of the character input stream

ever, and some other protection scheme is necessary. Covering the password with noise characters is the usual solution.

The HOST-HOST protocol provides a large number of sockets for each HOST, but carefully refrains from specifying which ones are to be used for what. To establish communication between a user-TELNET and a server-TELNET some convention is required. The Initial Connection Protocol (ICP)[10] is used:

1. Connection is initiated from a user-TELNET's receive socket to a serving HOST's socket 1 (a send socket).
2. When the initial connection is established, the serving HOST sends a generated socket number and closes the connection. This socket number identifies an adjacent socket pair at the serving HOST through which the user-TELNET can communicate with a server-TELNET.
3. TELNET connections are then initiated between the now specified pairs of sockets. Two connections are used to provide bi-directional communication.

Note that socket 1 at the serving HOST is in use only long enough to send another socket number with which to make the actual service connections.

One of the functions performed by a terminal control program within an operating system is the scanning of an input stream for attention characters intended to stop an errant process and to return control to the executive. Terminal control programs which buffer input sometimes run out of space. When this happens to a local terminal's input stream, a "bell" or a question mark is echoed and the overflow character discarded, *after checking to see if it is the attention character*. See Figure 7. This strategy works well in practice, but it depends rather strongly on the intelligence of the human user, the invariant time delay in the input transmission system, and a lack of buffering between type-in and attention checking. None of these conditions exists for interactive traffic over the net: The serving HOST cannot control the speed (except to slow it down) or the buffering within the using HOST, nor can it even know whether a human user is supplying the input. It is thus necessary that the terminal control program or server-TELNET not, in general, discard characters from a network input stream; instead it must suspend its acceptance of characters via the HOST-HOST flow control mechanism. Since a HOST may only send messages when there is room at the destination, the responsibility for dealing with too much input is thus transferred back to the using HOST. This scheme assures that no characters accepted by the using HOST are inadvertently lost. However, if the process in the serving HOST stops accepting input, the pipeline of buffers between the user-TELNET and remote process will fill up so that attention characters cannot get through to the serving executive. In the TELNET protocol, the solution to this problem calls for the user-TELNET to send, on request, a HOST-HOST interrupt signal forcing the server-TELNET to switch input modes to process network input for attention characters. The server-TELNET is required to scan for attention characters in its network input, even if some input must be discarded while doing so. The effect of the interrupt signal to a server-TELNET from its user is to cause the buffers between them to be emptied for the priority processing of attention characters.

To flip an attention scanning server-TELNET back into its normal mode, a special TELNET synchronization character (SYNCH) is defined. When the server-TELNET encounters this character, it returns to the strategy of accepting terminal input only as buffer space permits. There is a possible race condition if the SYNCH character arrives before the HOST-HOST interrupt signal, but the race is handled by keeping a count of SYNCHs without matching signals. Note that attention characters are HOST specific and may be any of 129 characters—128 ASCII plus "long space"—while SYNCH is a TELNET control character recognized by all server-TELNETs. It would not do to use the HOST-HOST signal alone in place of the signal-SYNCH combination in attention processing, because the position of the SYNCH character in the TELNET input stream is required to determine where attention processing ends and where normal mode input processing begins.

FILE TRANSFER

When viewing the ARPANET as a distributed computer operating system, one initial question is that of how to construct a distributed file system. Although it is constructive to entertain speculation on how the ultimate, automated distributed file system might look, one important first step is to provide for the simplest kinds of explicit file transfer to support early substantive use.

During and immediately after the construction of the ARPANET user-level process interface, several *ad hoc* file transfer mechanisms developed to provide support for initial use. These mechanisms took two forms: (1) use of the TELNET data paths for text file transfer and (2) use of raw byte-stream communication between compatible systems.

By adding two simple features to the user-TELNET, text file transfer became an immediate reality. By adding a "SCRIPT" feature to user-TELNETS whereby all text typed on the user's console can be directed to a file on the user's local file system, a user need only request of a remote HOST that a particular text file be typed on his console to get that file transferred to his local file system. By adding a "SEND-FILE" feature to a user-TELNET whereby the contents of a text file can be substituted for console type-in, a user need only start a remote system's editor as if to enter new text and then send his local file as type-in to get it transferred to the remote file system. Though crude, both of these mechanisms have been used with much success in getting real work done.

Between two identical systems it has been a simple matter to produce programs at two ends of a connection to copy raw bits from one file system to another. This mechanism has also served well in the absence of a more general and powerful file transfer system.

Ways in which these early *ad hoc* file transfer mechanisms are deficient are that (1) they require explicit and often elaborate user intervention and (2) they depend a great deal on the compatibility of the file systems involved. There is an on-going effort to construct a File Transfer Protocol (FTP)[11,12] worthy of wide implementation which will make it possible to exchange structured sequential files among widely differing file systems with a minimum (if any) explicit user intervention. In short, the file transfer protocol being developed provides for the connection of a file transfer user process ("user-FTP") and file transfer server process ("server-FTP") according to the Initial Connection Protocol discussed above. See Figure 8. A user will be able to request that specific file manipulation operations be performed on his behalf. The File Transfer Protocol will support file operations including (1) list remote directory, (2) send local file, (3) retrieve remote file, (4) rename remote file, and (5) delete remote file.

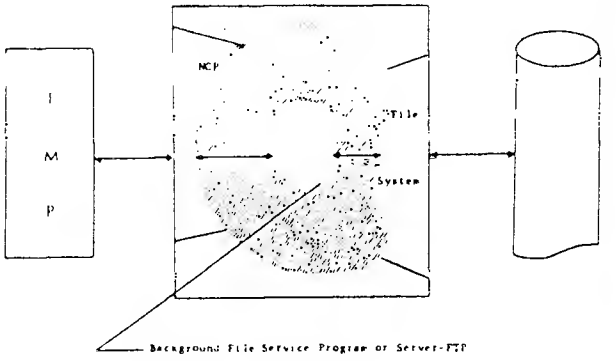


Figure 8—Data flow for file transfer

It is the intention of the protocol designers to regularize the protocol so that file transfer commands can be exchanged by consoles file transfer jobs engaged in such exotic activities as automatic back-up and dynamic file migration. The transfers envisioned will be accompanied with a Data Transfer Protocol (DTP)[11] rich enough to preserve sequential file structure and in a general enough way to permit data to flow between different file systems.

USING THE ARPANET FOR REMOTE JOB ENTRY

A very important use of the ARPANET is to give a wide community of users access to specialized facilities. One type of facility of interest is that of a very powerful number-cruncher. Users in the distributed ARPANET community need to have access to powerful machines for compute-intensive applications and the mode of operation most suited to these uses has been batch Remote Job Entry (RJE). Typically, a user will generate a "deck" for submission to a batch system. See Figure 9. He expects to wait for a period on the order of tens of minutes or hours for that "deck" to be processed, and then to receive the usually voluminous output thereby generated. See Figure 10.

As in the case of file transfer, there are a few useful *ad hoc* ARPANET RJE protocols. A standard RJE protocol is being developed to provide for job submission to a number of facilities in the ARPANET. This protocol is being constructed using the TELNET and File Transfer protocols. A scenario which sketches how the protocol provides the RJE in the simplest, most explicit way is as follows:

Via an ARPANET RJE process, a user connects his

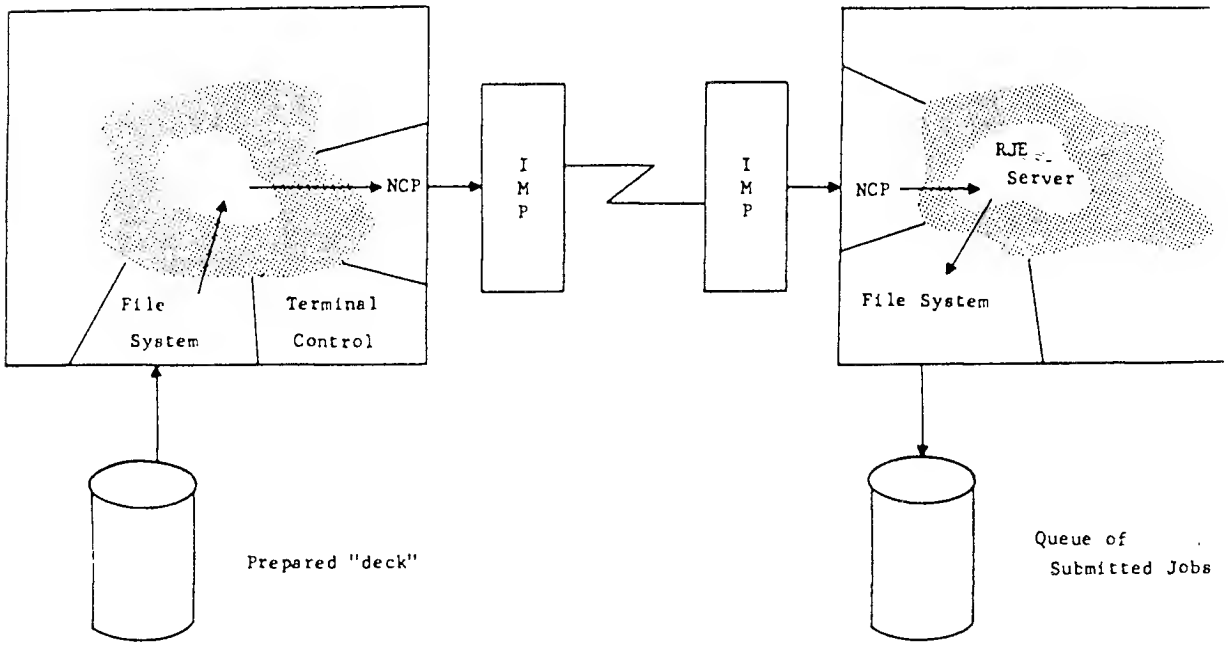


Figure 9—Submission of RJE input

terminal to an RJE server process at the HOST to which he intends to submit his job "deck." Through a short dialogue, he establishes the source of his input and initiates its transfer using the File Transfer Protocol. At some later time, the user reconnects to the appropriate RJE server and makes an inquiry on the

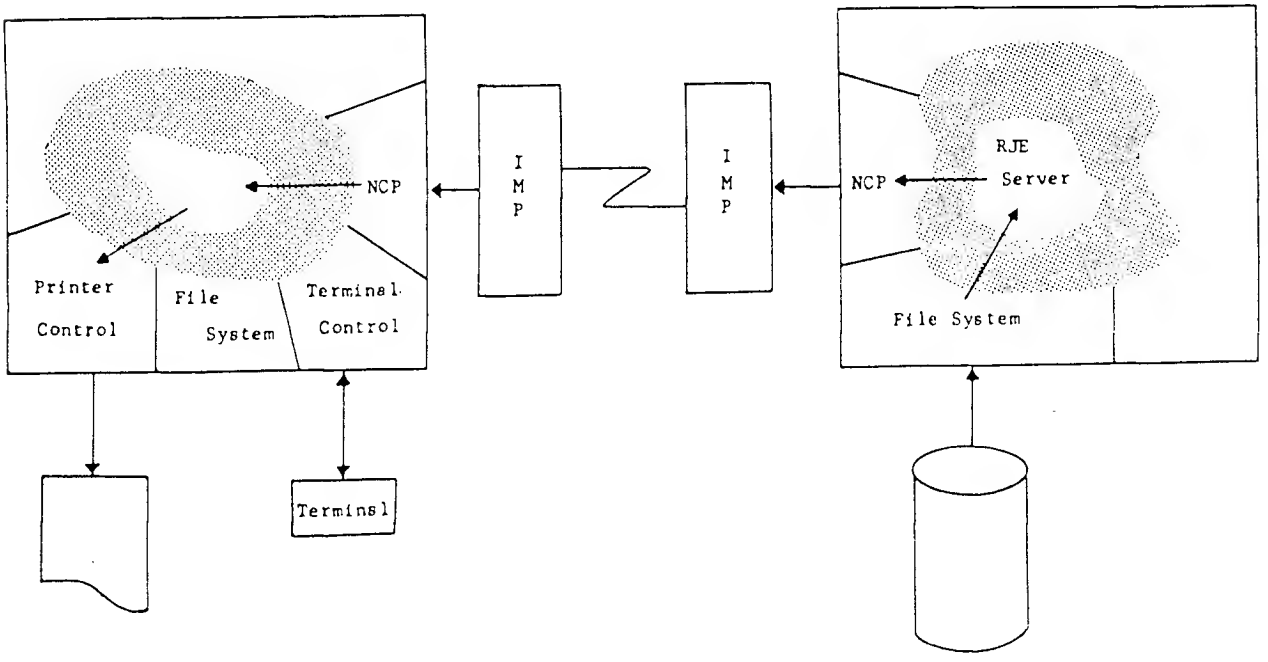


Figure 10—Retrieval of RJE output

status of his job. When notified that his input has been processed, he then issues commands to the serving HOST to transfer his output back.

We can of course imagine more automatic ways of achieving these same functions. A user might need only type a job submission command to his local system. Automatically and invisibly, then, the local system would connect and converse with the specified RJE server causing the desired output to later appear in the users file area or perhaps on a local line printer. The intention is to design the RJE protocol so that the explicit use can start immediately and the more automatic RJE systems can be built as desired.

OTHER PROTOCOLS AND CONCLUSIONS

One of the more difficult problems in utilizing a network of diverse computers and operating systems is that of dealing with incompatible data streams. Computers and their language processors have many ways of representing data. To make use of different computers it is necessary to (1) produce a mediation scheme for each incompatibility or (2) produce a standard representation. There are many strong arguments for a standard representation, but it has been hypothesized that if there were a simple way of expressing a limited set of transformations on data streams, that a large number of incompatibilities could be resolved and a great deal of computer-computer cooperation expedited.

The bulk of protocol work is being done with the invention of standard representations. The TELNET protocol, as discussed, is founded on the notion of a standard terminal called the Network Virtual Terminal (NVT). The File Transfer Protocol is working toward a standard sequential file (a Network Virtual File?). So it is also with less advanced protocol work in graphics and data management.

There is one experiment which is taking the transformational approach to dealing with incompatibilities. The Data Reconfiguration Service (DRS) is to be generally available for mediating between incompatible stream configurations as directed by user-supplied transformations.[13]

ACKNOWLEDGMENTS

Function-oriented protocols have been the principal concern of the ARPANET Network Working Group (NWG). A list of people who have contributed to the development of the protocols discussed would include, Robert Braden, Howard Brodie, Abhay Bhushan, Steve Carr, Vint Cerf, Will Crowther, Eric Harslem, Peggy Karp, Charles Kline, Douglas McKay, Alex McKenzie, John Melvin, Ed Meyer, Jim Michener, Tom O'Sullivan, Mike Padlipsky, Aric Shoshani, Bob Sundberg, Al Vezza, Dave Walden, Jim White, and Steve Wolfe. We would like to acknowledge the contribution of these researchers and others in the ARPA Network Working Group, without assigning any responsibility for the content of this paper.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings May 1970

2 F E HEART et al
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings May 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings May 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA Computer network
AFIPS Conference Proceedings May 1970

5 *Specifications for the interconnection of a Host and an IMP*
Bolt Beranek and Newman Inc Report No 1822
February 1971

6 C S CARR S D CROCKER V G CERF
HOST-HOST communication protocol in the ARPA Network
AFIPS Conference Proceedings May 1970

7 *HOST/HOST protocol for the ARPA Network*
ARPA Network Information Center #7147

8 T O'SULLIVAN et al
TELNET protocol
ARPA Network Working Group Request For Comments (RFC) #158 ARPA Network Information Center (NIC) #6768 May 1971

9 S M ORNSTEIN et al
The Terminal IMP for the ARPA Computer Network
AFIPS Conference Proceedings May 1972

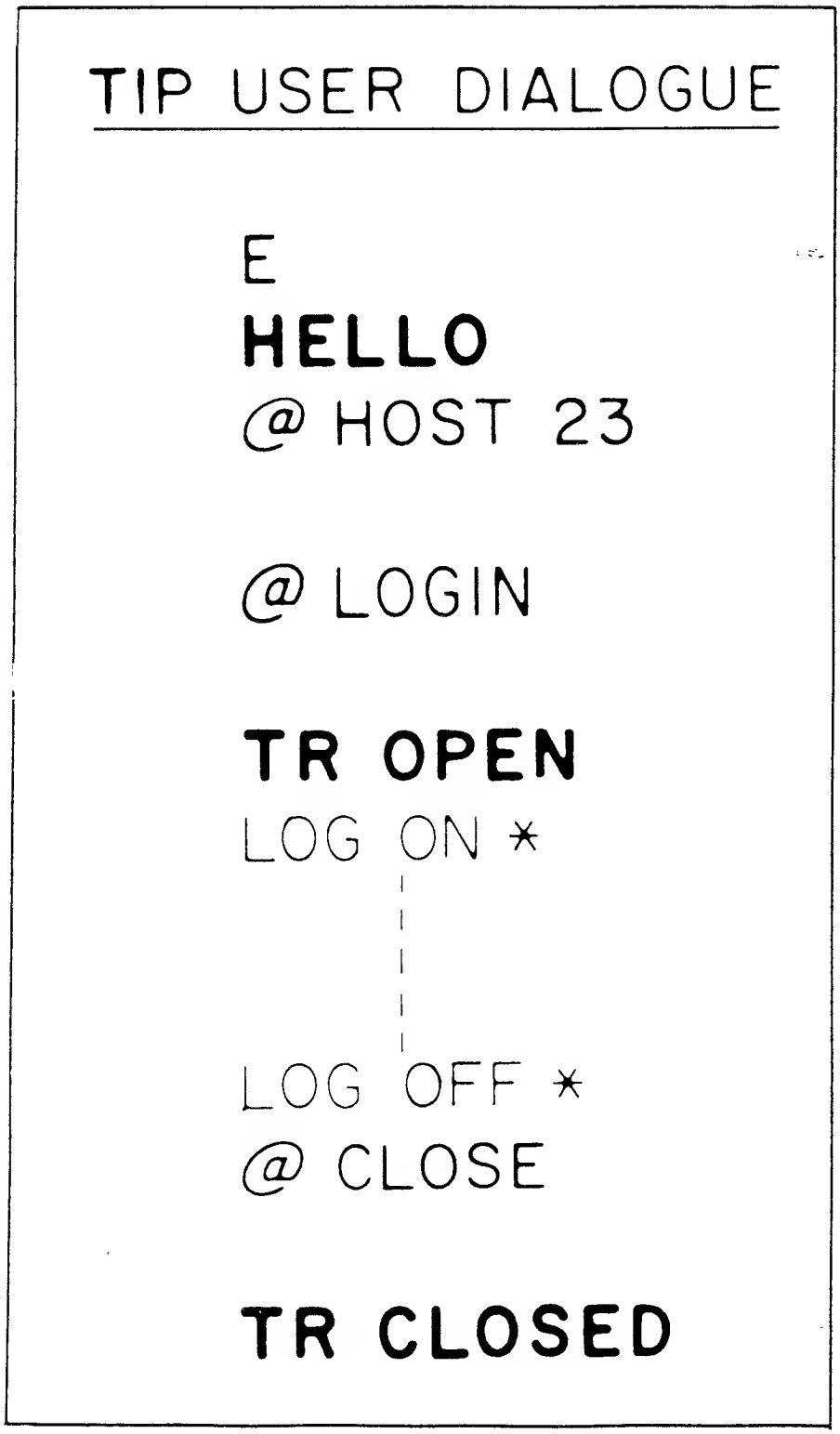
10 J B POSTEL
Official initial connection protocol
ARPA Network Working Group Document #2 NIC #7101 June 1971

11 A K BHUSHAN et al
The data transfer protocol
RFC #264 NIC #7812 November 1971

12 A K BHUSHAN et al
The file transfer protocol
RFC #265 NIC #7813 November 1971

13 R ANDERSON et al
The data reconfiguration service—An experiment in adaptable process/process communication
The ACM/IEEE Second Symposium On Problems In The Optimization Of Data Communications Systems
October 1971

[redacted]


TIP USER DIALOGUE
E
HELLO
@ HOST 23
@ LOGIN
TR OPEN
LOG ON *
LOG OFF *
@ CLOSE
TR CLOSED

McROSS—A multi-computer programming system*

by ROBERT H. THOMAS and D. AUSTIN HENDERSON

Bolt, Beranek and Newman, Inc.
Cambridge, Massachusetts

INTRODUCTION

This paper describes an experimental "distributed" programming system which makes it possible to create multi-computer programs and to run them on computers connected by the ARPA computer network (ARPANET).[1] The programming system, which is called McROSS (for **M**ulti-Computer **R**oute **O**riented **S**imulation **S**ystem), is an extension of a single-computer simulation system for modelling air traffic situations[2] developed by Bolt, Beranek and Newman, Inc. (BBN) as a tool for air traffic control research. The McROSS system provides two basic capabilities. One is the ability to program air traffic simulations composed of a number of "parts" which run in geographically separated computers, the distributed parts forming the nodes of a "simulator network." The second is the ability of such a simulator network to permit programs running at arbitrary sites in the ARPANET to "attach" to particular nodes in it for the purpose of remotely monitoring or controlling the node's operation.

The McROSS distributed programming system is unique in several ways:

(a) Use of McROSS generates inter-computer traffic in which a group of programs are engaged in substantive conversation. There is relatively little previous experience with such inter-computer, program-to-program conversations.

(b) The component nodes of a simulator network are not bound to particular ARPANET sites until simulation "run time." Thus on different runs the same distributed program can be distributed in different ways over the ARPANET. For example, in one run all the nodes of a simulator network might be run at BBN and on the next some might be run at BBN, others at RAND and still others at the University of Utah. This mode of using the ARPANET is significantly different from the normal one in which programs are bound to particular network sites at program composition time. (The only constraint on the binding of nodes to ARPANET sites is the requirement that each node run on a PDP-10 under the TENEX operating system.[5])

(c) The responsibilities of a node in a simulator network can be conveniently partitioned into subtasks which can be performed more or less independently of one another. The McROSS implementation mirrors this partitioning. Functions performed at the nodes are realized by groups of loosely connected, concurrently evolving processes.

The distributed simulation system represents an initial step in an on-going research program which is investigating techniques to make it easy to create, run and debug computations involving the coordinated behavior of many computers. The McROSS system is intended to serve both as an experimental vehicle for studying problems related to distributed computation and as a tool for air traffic control research. Its two goals are well matched. A satisfactory solution to the nation's air traffic problems is likely to include a network of airborne and ground based computers working together on a single distributed computation: the scheduling and control of aircraft maneuvers. Thus, the air traffic control problem is a rich source of interesting problems in partitioned computation which can be used to measure the usefulness of the distributed computational techniques developed.

This paper is a report on one phase of a continuing research program. The intent is to describe interesting aspects of an experimental distributed programming

* This work was supported by the Advanced Projects Research Agency of the Department of Defense under Contract No. DAHC-71-C-0088.

system and to share our experience with others considering the construction of such systems. The paper does no more than hint at how the McROSS system can be used in air traffic control research.

The next section provides background useful for subsequent discussion of the distributed programming system. After than, the McROSS system is described, first in terms of the facilities it provides for creating distributed simulations, and then in terms of interesting aspects of its implementation. The paper closes with a discussion of some of the issues brought into focus by the experience of implementing and using a distributed programming system.

COMPONENTS OF THE McROSS SYSTEM

The main components of the distributed programming system are:

1. The ARPA computer network;
2. The TENEX operating system for the PDP-10; and
3. A simulation programming system known as ROSS (for **R**oute **O**riented **S**imulation **S**ystem).

These components became operational within 6-10 months of one another, at which time it became feasible to implement the distributed programming system being described.

The ARPANET

The ARPA computer network[1,3,4] is a set of autonomous computer systems (*hosts*) which are interconnected to permit resource sharing between any pair of them. The goal of the ARPANET is for each host to make its resources accessible from other hosts in the net thereby permitting persons or programs residing at one network site to use data and programs that reside and run at any other. Each host interfaces with the network through an Interface Message Processor (*IMP*),[3] a small dedicated general-purpose computer. The IMPs, which reside at the various ARPANET sites, are connected by 50 kilobit common carrier lines and are programmed to implement a store and forward communication network. In passing from host A to host B a message passes from host A to IMP A, through the IMP communication network from IMP A to IMP B (in general passing through a number of intermediate IMPS) and finally from IMP B to host B.

At each host there is a Network Control Program (*NCP*) whose function is to provide an interface between the network and processes within the host. The NCPs use the IMP store and forward network to provide processes with a connection switching network. Thus, they enable processes in different hosts to establish connections with one another and to exchange information without directly concerning themselves with details of network implementation such as the way in which hosts are connected to and communicate with IMPs.

Information can flow in only one direction on an ARPANET *connection*. Thus, before two processes are able to engage in a dialogue they must establish two such connections between them. A connection is completely specified by its two ends which are called *sockets*. A network socket is itself uniquely specified by a host identifier and a socket identifier. The purpose of the socket identifier is to specify a process or group of processes within a host and a socket relative to that process or process group. Thus, it is useful to think of a socket as having a three component "socket name" of the form H.P.N. H is the "host" component which identifies an ARPANET host, P is the "process" component which identifies a process group within H and N is the "process-local" component which identifies a socket relative to P. In the sequel

$$H_1.P_1.N_1 \rightarrow H_2.P_2.N_2$$

is used to denote a connection between sockets $H_1.P_1.N_1$ and $H_2.P_2.N_2$ where $\rightarrow$ indicates the direction of information flow over the connection.

For a connection to be established between two processes each must request that it be established. There are two common ways in which connections are established. In the first, the processes play symmetric roles. Each specifies, as part of a "request for connection" (*RFC*), the socket name at its end of the connection, the socket name at the remote end of the connection and the direction of information flow. If the two RFCs "match" the connection is established. This connection procedure requires *a priori* agreement upon the sockets to be used for the connection. The second common connection procedure is used in situations in which one process wishes to provide some sort of service to other processes. The "serving" process establishes a *listening* socket within its host which is receptive to RFCs from any other process in the network and then "listens" for connection attempts. The serving process uses facilities provided by its NCP to detect the occurrence of a connection attempt and to determine its source. When such an attempt is made the serving process can choose to accept or reject it. This connection procedure requires that only one socket name, that of the serving process's listening socket, be known

a priori. In the remainder of this paper

$$[H.P.N{\rightarrow}]_L$$

and

$$[H.P.N{\leftarrow}]_L$$

are used to denote connections established in a listening state.

The TENEX operating system

TENEX[4] is a time sharing system for the DEC PDP-10 processor augmented with paging hardware developed at BBN. For purposes of this paper it is useful to describe TENEX in terms of the virtual processor it implements for each logged-in user (i.e., user time sharing job).

The instruction repetoire of the TENEX virtual processor includes the PDP-10 instruction set with the exception of the direct I/O instructions. In addition, it includes instructions which provide access to virtual processor capabilities implemented by the combination of the TENEX software and hardware.

The TENEX virtual processor permits a user job to create a tree-structured hierarchy of processes. Such processes have independent memory spaces and computational power. At different stages in its lifetime a single user job may include different numbers of processes in various states of activity. Several mechanisms for interprocess communication are provided by the TENEX virtual machine. Processes can interact by sharing memory, by interrupts and by direct process control (e.g., one process stops, modifies and restarts another which is "inferior" to it in the hierarchy).

A memory space is provided by the virtual processor which is independent of the system configuration of core memory. Each process has a memory space of 256K words which is divided into 512 pages each of 512 words. A process can specify read, write and execute protection for pages in its memory space as it sees fit.

The virtual machine includes a file system which provides a mechanism for storing information on and retrieving it from external devices attached to TENEX. Processes refer to files using symbolic names, part of which identifies the particular device on which the file resides. The instruction set of the virtual machine includes operations for data transfer to and from files which a process can execute without explicitly arranging for buffering.

The NCP resident in TENEX makes the ARPANET appear to TENEX processes as an I/O device. The name of a "file" corresponding to a network connection includes the names of both the local socket and the remote socket which define the connection. A process requests TENEX to establish a connection by attempting to open an appropriately named "network" file. The open attempt succeeds and the connection is established if and when another process issues a matching RFC. TENEX processes transmit and receive information over network connections by executing the normal file data transfer instructions.

ROSS

ROSS is a programming system for modelling air traffic situations.[2] It includes facilities for creating and running simulation experiments involving air traffic in an experimenter-defined airspace. The system is currently being used in a number of air traffic control research projects.

To create a simulation experiment the experimenter uses ROSS language forms to write a "program" which defines the geography of an airspace, the wind profile for the airspace, aircraft flight characteristics and a set of "route procedures." A *route procedure* consists of a sequence of one or more "instructions" for monitoring or controlling the progress of an aircraft through the simulated airspace. Execution of a route procedure is the ROSS counterpart of pilot and/or controller actions. The "flight" of a simulated aircraft through the airspace is accomplished by the execution of a series of route procedures. ROSS includes "primitive" route procedures to "create" aircraft, "inject" them into and remove them from the simulated airspace as well as ones which cause aircraft to accelerate, turn, climb and descend.

By compiling his program the experimenter creates a simulator for traffic in the airspace he has defined. To perform a simulation experiment the experimenter runs his program. The simulated airspace remains empty until the program is supplied with input which inject aircraft into the airspace and control their flight through it. Each input line the simulator receives is passed to an internal parsing mechanism which issues a call to an appropriate experimenter-defined or primitive route procedure. The program can accept input from an on-line terminal, a predefined "scenario" file, or both. Input lines from the scenario file are identical to ones from the on-line keyboard with the exception that scenario file input lines include a time field which specifies the (simulated) time at which the program is to accept them. A user can manually "vector" aircraft through the airspace by supplying input at the on-line keyboard.

A ROSS simulator can drive displays of the airspace

and, in addition, can generate output which can be written into files, typed on the on-line keyboard or sent back into the simulator input parser.

THE McROSS PROGRAMMING SYSTEM

The McROSS system provides the ability to define simulation experiments involving a number of airspaces or "simulation centers" which can be interconnected to form a simulator network. Adjacent centers in the simulator network are connected to one another by way of the ARPANET. The components of a simulator network may run as user jobs distributed among different TENEXs or as different user jobs on the same TENEX. (As of January 1972 the ARPANET included five TENEX hosts.)

Computational responsibility for performing a multi-computer McROSS simulation is truly distributed. For example, as an aircraft flies from one airspace into an adjacent one the responsibility for simulating its dynamics shifts from one computer to another.

Goals

The McROSS system was implemented to achieve the following goals:

1. Autonomy of parts:
   Individual components of a McROSS network should be able to operate independently of one another (to the extent that each is independent of the others for traffic). Furthermore, no center should be able to cause another to malfunction. Autonomy of parts enables a multi-computer simulation to run when only some of its components are operational. Failure of a component center in a multi-center simulation results in degradation of the total simulation rather than forced termination of it. A beneficial side effect of autonomy is that individual centers can be partially debugged without running the entire simulator network.
2. Deferral of process/processor binding:
   The binding of centers in a McROSS network to host computers in the ARPANET should be deferred until run time. This goal can be stated in more general terms. The program for a distributed computation defines a logical configuration made up of abstract relations between the computation's parts. A given execution of the program is accomplished by a particular physical configuration of computers. The two configurations, logical and physical, are by necessity related. However, the programmer should have the option of specifying them separately. By deferring process/processor binding the configurations can be separately specified. As a result the programmer is free while composing his program to concentrate on the logical structure he wants his program to define without concerning himself with details of the physical structure on which it is to be run.
3. Capability for dynamic reconfiguration:
   In the course of a simulation it should be possible for adjacent centers to dynamically break and reestablish connections with one another. Furthermore, it should be possible for process/processor binding to be changed during a simulation. That is, it should be possible to change the physical location of a center from one ARPANET host to another. The ability to dynamically reconfigure makes it possible to remove an improperly operating center from the simulator network and replace it with another at the same ARPANET host or at a different one.
4. Decentralization of control:
   McROSS is to be used as a tool for investigating distributed computation. Among the subjects to be studied are methods for controlling such computations. In particular, various techniques for distributing control responsibilities among the parts of a computation are to be experimentally investigated. It is important, therefore, that operation of the McROSS system not require a central control mechanism for coordinating simulator networks. Stated somewhat differently, the only components required for a McROSS simulation should be simulation centers defined by McROSS programs. The realization of this goal, which makes experimentation with distributed control possible, should not preclude experimentation with centralized control.
5. Remote monitoring capability:
   A McROSS simulator network should provide ports through which its components are accessible to any ARPANET host. An appropriately programmed process running at any ARPANET host should be able to "attach" to a component of a simulator network to monitor and control its operation. A remote monitoring process should be able to:

   a. obtain sufficient information to display traffic in the airspace it is monitoring.
   b. serve as the on-line keyboard for the center it is monitoring;

c. "detach" from one center and "attach" to another in the course of a simulation run.

McROSS as seen by the user

A McROSS simulator network is defined by a program composed of a "network geometry" sub-program and sub-programs corresponding to each of the centers in the network.

The network geometry sub-program defines the logical geometry for a simulator network. Conceptually, a network is composed of nodes and arcs. Nodes in a McROSS network are simulation centers and arcs are duplex connections between centers. Figure 1 shows a four node simulator network which could be used to simulate air traffic between Boston and New York. The following geometry sub-program defines that network:

```
netbegin
    netcen BOSTRM, BOSCEN, NYCEN,
           NYTRM
    netcon BOSTRM, BOSCEN
    netcon BOSCEN, NYCEN
    netcon NYCEN, NYTRM
netend
```

The *netcen* statement declares that the network contains four nodes (Boston terminal control, Boston en route control, New York en route control and New York terminal control). The *netcon* statements declare the three arcs; *netbegin* and *netend* serve to bracket the geometry declarations.

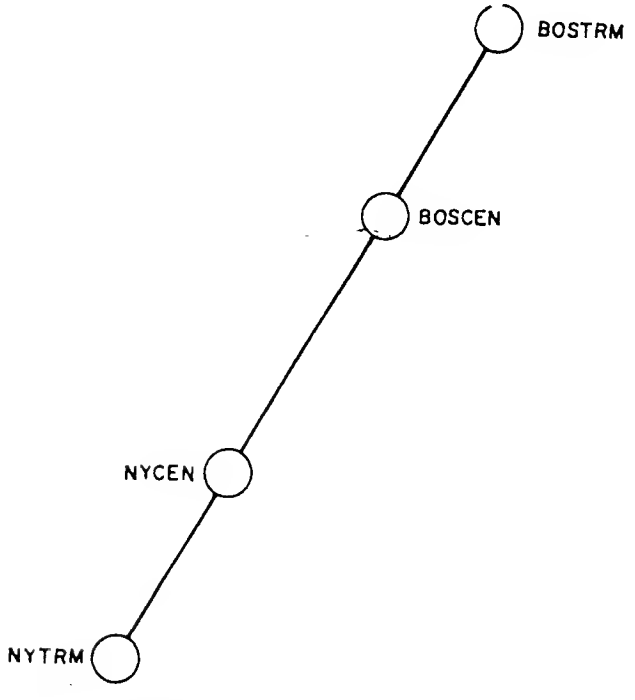


Figure 1—A simulator network which could be used to simulate air traffic between Boston and New York

In general, the sub-program for each center has four parts:

- a route procedure module
- a local geography module
- a wind profile module
- an aircraft characteristics module.

In addition to defining procedures followed by aircraft as they fly through a center's airspace, the route procedure module includes routines specifying how the center interacts with its neighbors. Information exchange between adjacent centers is accomplished by sending messages across the connection between them. A center handles messages from neighboring centers by submitting them to its input parsing mechanism. Such messages are treated identically to input from its on-line console and scenario file.

The ability of adjacent centers to interact depends upon the state of the connection between them as each sees it from its end of the connection. A center may consider a connection to be in one of three states:

1. uninitialized:
   the "physical location" of the neighbor is unknown;
2. closed:
   the "physical location" of the neighbor is known but the connection is not capable of carrying messages (either because it has not yet been established or because it has been broken);
3. open:
   the connection may be used for information exchange with the neighbor.

In the current implementation of McROSS the "physical location" of a neighbor includes both the ARPANET host the center is running on (e.g., BBN) and the identification of the user it is running under (e.g., Jones).

McROSS provides the operations *init*, *conn*, *dsconn* and *abort* for changing the state of a connection. The effect these operations have on the end of a connection is illustrated by the state transition diagram of Figure 2.

Consider the geometry for the Boston-New York simulation. Execution of

```
conn BOSTRM
```

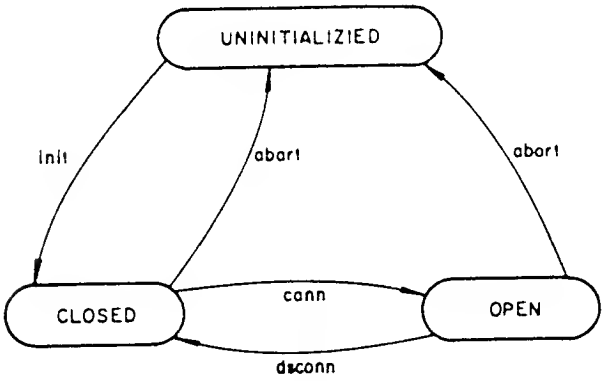


Figure 2—Transition diagram for the state of the end of a connection showing the effect of the operations *init*, *conn*, *dsconn* and *abort*

within the BOSCEN initiates an attempt to open a connection with BOSTRM. The connection attempt succeeds if a matching *conn* is executed by the BOSTRM center. The effect of executing

dsconn NYCEN

within the BOSCEN center simulator is to break the connection between the NYCEN and BOSCEN centers by forcing both ends of it into the closed state; *abort* works in an analogous manner. Execution of the *init* operation results in a center-user dialogue in which the human user is asked by the center program to specify the physical location of the neighbor.

Two language primitives are provided for sending messages from one center to another. One takes a single operand, a message, which it sends to every neighbor whose connection is in the open state. The other takes two operands, a message and the name of a neighboring center. If the connection to the center is open, the message is sent to the center; otherwise the operation has no effect. Because they are submitted to the input parsing mechanism care must be taken that messages sent to a neighbor are formatted correctly.

McROSS includes operations which can be used by a center to obtain information about the simulator network and its immediate neighbors. For example, there is a primitive which produces a list of all nodes in the network (i.e., all centers declared by the *netcen* declaration); another one produces a list of all neighboring centers for which connections are in the open state. In addition, a center can examine the state of the connection to each of its neighbors individually.

The local geography module defines the airspace of a center by specifying names and locations (x-y coordinates) of important geographic features such as navigational aids, obstructions and airports. In addition it includes a declarative statement which names the simulation center. For example, the geography module for the BOSTRM center would include the declaration

atccen BOSTRM.

This declaration has the effect of binding the identifier THISIS to the name BOSTRM. Thus in the BOSTRM center THISIS is bound to BOSTRM while in the NYTRM center it is bound to NYTRM. Route procedures which access the simulator center name can be written in terms of THISIS to enable their use at any center in a simulator network.

A properly authorized process at any ARPANET host can attach to a center in a McROSS simulator network and request output from it and direct input to it thereby monitoring and controlling its operation. McROSS center programs are prepared to offer two kinds of service to remote monitors:

1. broadcast service:
   Centers continuously broadcast certain information to monitors attached to them. Presently centers broadcast flight parameters of each aircraft in their air space (speed, heading, altitude, x-position, y-position, acceleration, aircraft id) and the (simulated) time. A remote monitor can use broadcast information to drive a display of traffic in a center's airspace or it can record it for later analysis.
2. demand service:
   Each center is prepared to respond to certain specific requests from monitors. In the current implementation a monitor can request that a center:
   a. transmit its map of the airspace (which can be used as background for displaying the center's air traffic);
   b. stop the continuous broadcast;
   c. resume the continuous broadcast;
   d. treat the monitor as its on-line keyboard by directing keyboard output to the monitor and accepting input from it;
   e. cease treating the monitor as its on-line keyboard;
   f. break its connection with the monitor.

The monitoring facility has proven useful both for debugging and for demonstration purposes. One difficulty a user faces in debugging a multi-center simulation is determining what is happening at centers suspected to be malfunctioning. A monitor, constructed appropriately, can serve as a "graphical probe" and be

used to watch the operation of first one suspect center and then another. For example, we have used such a monitor to follow the trajectory of an aircraft as it passes through several centers.

By enabling processes at arbitrary ARPANET sites to observe and control McROSS simulations, the monitoring facility provides a mechanism for using hardware and software features which are unique to various network installations. By using monitors which play an active role in his simulations a McROSS user can experiment with different ways of partitioning computational and control responsibilities in air traffic situations. He could, for example, experiment with a monitor built to provide a weather advisory service for simulator centers. Such a monitor would presumably have access to an on-line weather data base. (A weather data base to be accessible through the ARPANET is currently being designed.[10]) To perform its service for a center the monitor would attach to the center, requesting that the center broadcast aircraft flight parameters to it and accept input lines from it. It would then "watch" the airspace of the center and send instructions to it, as necessary, to vector aircraft around severe weather.

Unless he chooses to do so, the simulation programmer need not concern himself with remote monitoring beyond specifying at simulation run time which centers in his network are to be receptive to remote monitors. Monitors themselves are not part of the McROSS system. McROSS merely provides a mechanism for remote monitoring. No effort has been made to provide linguistic features within the McROSS system to make it easy to write programs to do monitoring.

THE McROSS IMPLEMENTATION

Some interesting aspects of the McROSS implementation are discussed in this section. The section focuses on strategy rather than detail. The result is a simplified but nonetheless accurate sketch of the implementation.

The ROSS implementation

Implementation of McROSS was accomplished by extending the existing ROSS simulation system. A ROSS simulation consists of initialization, simulation and termination phases. The simulation phase is implemented as a loop. Each pass through the loop represents a "tick" of the clock which maintains simulation time. On each tick the simulator:

1. parses and interprets input directed to it from its scenario file and on-line console since the last tick;
2. interprets the route procedure each aircraft is currently following;
3. advances each aircraft along its trajectory taking into account the aircraft's speed, acceleration and heading and the wind profile of the airspace;
4. directs output generated since the last tick to the appropriate devices;
5. performs actions necessary to maintain the local display of its airspace;
6. increments the simulated time.

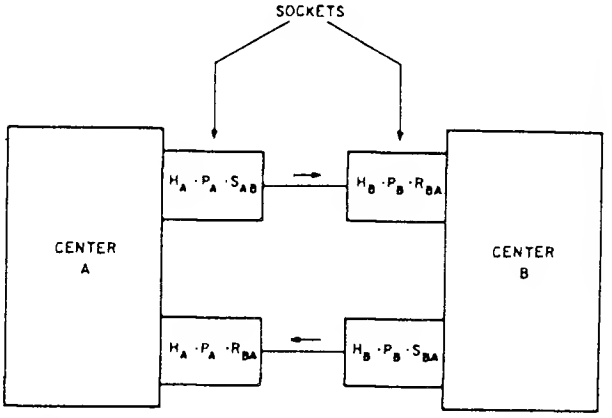


Figure 3—Schematic of center-center connection between adjacent centers A and B. S_{AB}, R_{AB}, S_{BA} and R_{BA} are assigned at program translation time; H_A, H_B, P_A and P_B are determined at run time

A ROSS simulation can be run either in a "real time" mode in which simulated time is locked to real time, or in a "hyper fast" mode in which the simulation proceeds as fast as it can.

Process/processor binding and center-center connections

Connections between pairs of simulator network centers are duplex. Each center-center connection is implemented by two ARPANET connections. More specifically, an open connection between centers A and B is realized by the two ARPANET connections (see Figure 3):

$$H_A.P_A.S_{AB} \rightarrow H_B.P_B.R_{BA}$$

$$H_A.P_A.R_{AB} \leftarrow H_B.P_B.S_{BA}$$

To establish such a center-center connection two pairs of matching RFCs must be issued by centers A and B. To issue the correct RFCs A must know H_B, P_B, R_{BA} and S_{BA}; similarly, B must know H_A, P_A, R_{AB} and S_{AB}.

The host (H) and process (P) components of the socket names for a center-center connection cannot be determined until run-time because process/processor binding is deferred until then. However, the process-local components (R and S) of the socket names can be pre-assigned and, in fact, the effect of the declarations in the network geometry sub-program for a particular McROSS network is to do exactly that.

The process local components for the four socket names corresponding to a center-center connection are always the same whereas the host and process components may change from run to run or even within the same run if either neighbor is involved in a dynamic reconfiguration.

When a center's end of a center-center connection is in the uninitialized state the host and process components of the socket names corresponding to the remote end are unknown to it. To move its end of a connection from the uninitialized state the center engages in a dialogue with the user requesting from him the physical location of the neighbor. After successfully completing the dialogue the center has sufficient information to issue the two RFCs required of it to establish the connection.

Center-monitor connections

The connection between a center C and an attached remote monitor M is realized by two ARPANET connections. One of them

$$H_C.P_C.S_{CM} \rightarrow H_M.P_M.R_{MC}$$

is a "broadcast" connection used for continuously broadcasting information to M. The other

$$[H_C.P_C.R \leftarrow]_L$$

is a "request" connection maintained by C in a listening state for M to use to make requests of C. Each monitor attached to C has its own broadcast connection but may share a request connection with other monitors.

To make a request of C, M connects to the request socket, transmits its request over it and then closes the request connection, freeing it for use by other monitors. The action taken by C of course depends upon the request. If the request were for the display map, C would transmit the map data over the broadcast connection to M.

Before the RFCs required to establish a center-monitor connection between C and M can be issued C must know H_M, P_M and R_{MC} and M must know H_C, P_C, S_{CM} and R. To obtain the required information, M and C engage in a connection protocol which is similar to the "initial connection protocol" developed by the ARPANET network working group.[6]

Each center willing to service monitors maintains as an "attach" socket a send socket in a listening state (see Figure 4.a). The attach socket for C could be denoted

$$[H_C.P_C.A \rightarrow]_L$$

The process local component (A) of the name for attach sockets is the same for all centers and is "well-advertised." Therefore, if M knows the physical location of C it can issue an RFC for C's attach socket. The effect of such an RFC is to establish the connection (Figure 4.b)

$$H_C.P_C.A \rightarrow H_M.P_M.R_M$$

Upon detecting an RFC for its attach socket, C notes H_M and P_M and transmits S_{CM} and R over the connection. Next both C and M break the connection and

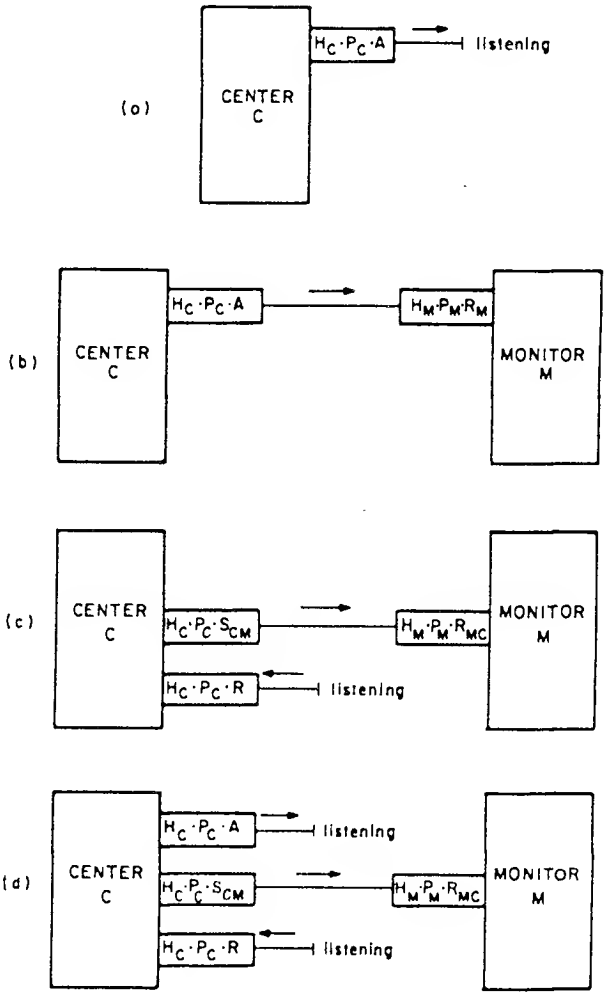


Figure 4—Schematic of connection protocol exchange between center C and monitor M

issue the RFCs necessary to establish the broadcast connection (Figure 4.c)

$$H_C.P_C.S_{CM} \rightarrow H_M.P_M.R_{MC}$$

where $R_{MC} = f(R_M)$, a pre-agreed upon function of R_M. C, if it has not done so previously for another monitor, sets up the listening connection

$$[H_C.P_C.R \leftarrow]_L$$

and finally, reestablishes its attach connection so that other monitors can attach to it (Figure 4.d). Currently the process-local components for ARPANET socket names are numbers and f is taken to be

$$f(x) = x+2.$$

If R_M rather than $f(R_M)$ were used for R_{MC} a race condition would result when M and C attempt to establish the broadcast connection. The race involves the order in which the connection with socket $H_M.P_M.R_M$ would be closed and reopened by C and M. In particular, the current TENEX NCP is such that an attempt by C to open the network file corresponding to its end of the (broadcast) connection

$$H_C.P_C.S_{CM} \rightarrow H_M.P_M.R_M$$

before M closes the network file corresponding to its end of the connection

$$H_C.P_C.A \rightarrow H_M.P_M.R_M$$

would fail. Use of $f(R_M)$ for R_{MC} avoids the race. A similar race condition, discovered in an early version of the ARPANET initial connection protocol,[7] is avoided in the current protocol by the same technique.[8]

Process structure at McROSS centers

A McROSS center is realized by a collection of cooperating, asynchronously evolving, sequential processes. The collection corresponds to a partitioning of the center's responsibilities into more or less independent sub-tasks. It includes:

1. a process (SIM) to perform the ROSS functions;
2. a process (CONN) for each center-center connection to establish and maintain the connection; and
3. a monitor server process (MONSER) to service remote monitors.

The CONN process at center A corresponding to the center-center connection to center B is responsible for establishing ARPANET send and receive connections with B. After it establishes the center-center connection

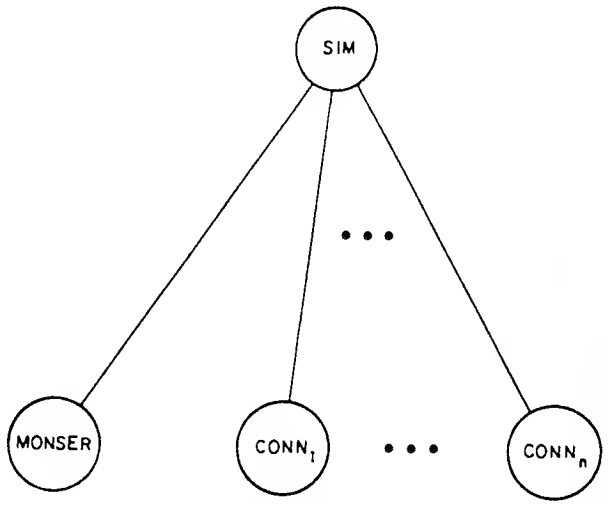


Figure 5—The process hierarchy which implements a McROSS simulation center with n neighbors

with B the CONN process maintains the connection. When messages from B arrive it passes them on to the SIM process for parsing.

The job of the MONSER process at a center is twofold: to engage in an initial connection protocol exchange with monitors attempting to attach to the center and to respond to requests made by attached monitors.

The processes at a center exist in a hierarchy (see Figure 5). The hierarchical structure is less the result of any one process being more important than any other than it is a consequence of the TENEX constraint that process groups be hierarchically arranged. During initialization the SIM process creates the MONSER and CONN processes. Thereafter, the processes evolve more or less independently.

The process structure at each center helps achieve autonomy of parts. The CONN processes and the MONSER process serve to protect the SIM process by insulating it from direct interaction with other centers and remote monitors.

Protocols

The current implementation of the TENEX NCP is such that if a center were to unilaterally break a connection with a neighbor (by closing the two corresponding ARPANET connections) it could leave processes in the neighboring center in an irrecoverable state. For example, a process in the neighbor sending information across the connection at the time it is broken would "feel" the close as if it had executed an illegal instruction. To prevent such situations McROSS centers en-

gage in a protocol exchange prior to breaking connections.

The center-center protocol is relatively simple. To perform an *abort* or *dsconn* operation a center sends its neighbor a "request for abort" or "request for disconnect" message and waits until it receives an acknowledgment before actually breaking the connection. Existence of the center-center protocol has two major implications. The first is that a center-center connection has more states than the three noted earlier. The additional states are transient ones which the connection passes through as the center and its neighbor advance through protocol exchanges initiated when one attempts to change the state of the connection. The transient states are invisible to the McROSS user. Immediately after a *dsconn* (or *abort*) is initiated the SIM process treats subsequent operations involving the connection as if the connection were already in the closed (or unini[illegible]ed) state. The second implication is that center-center connections carry "control" messages used in center-center protocol exchange in addition to "ordinary" messages intended for the receiver's parsing mechanism. The CONN process for each connection must be prepared to recognize and respond appropriately to both kinds of messages.

McROSS centers expect remote monitors to observe a center-monitor protocol. In addition to the connection and request procedures described earlier, the center-monitor protocol includes a disconnection procedure much like the one used in the center-center protocol.

Interprocess communication

To perform their tasks the processes at a simulation center must interact occasionally. For example, the arrival of a message from a neighbor requires interaction between the SIM and CONN processes. The CONN process receives the message and passes it onto the SIM process for parsing and interpretation.

One way the processes interact is through shared memory. For example, the SIM and CONN processes have read and write access to a shared "connection table." There is an entry in the table for each center-center connection which includes the state of the connection, a semaphore[9] and other information relevent to the connection. Use of the table is coordinated by strict adherence to a convention which requires every "critical" access (every write access and certain read accesses) to an entry to be bracketed by P (lock) and V (unlock) operations on the entry's semaphore.

The situation arising at a center when a neighbor attempts to break connection with the center is a typical one which requires interprocess communication. The CONN process corresponding to the center-center connection receives a "request for disconnect" message from the neighbor. Center-center protocol requires the CONN process acknowledge the request so that the neighbor can break the connection. The purpose of the protocol exchange is to warn the center that the connection is about to be broken and that it should no longer attempt to use it. Therefore before it acknowledges the neighbor's request it is important that the CONN process communicate this information to the other processes at its center. The CONN process does this by the following sequence of actions:

```
P(CONNECTION SEMAPHORE);
set connection-state to "not open";
V(CONNECTION SEMAPHORE);
acknowledge neighbor's request
```

As long as processes at the center perform the following sequence of actions to send over center-center connections there is no danger of sending after the connection has been closed:

```
P(CONNECTION SEMAPHORE);
if connection-state = open
    then send
    else abort the send;
V(CONNECTION SEMAPHORE)
```

Stated somewhat differently, sending over a center-center connection should be regarded as an operation which involves a "critical" read access of the corresponding connection table entry.

In addition to memory sharing, direct process control is used as a mechanism for interprocess communication in the McROSS system. Because of its superior position in the process hierarchy the SIM process can exert direct control over the other processes. A few situations occur in which it does so. For example, when a center-center connection has been closed or aborted (via *dsconn* or *abort*) the SIM process forces the corresponding CONN process to halt. If and when an attempt is initiated to reestablish the connection (via *conn*) SIM restarts it.

SOME OPEN QUESTIONS

This section briefly discusses questions representative of ones which have arisen in the course of using the McROSS system. The questions have been resolved to the extent that useful simulations can be performed using McROSS. However, none has been resolved in a

totally satisfactory manner. The intent of this section is to leave the reader with an appreciation for the issues raised by these questions; a thorough discussion of them is well beyond the scope of this paper.

Synchronization

Simulated time is important in the operation of the McROSS system. In particular, whenever an interaction between adjacent centers occurs it is important that the clocks kept by the centers show approximately the same time. Time synchronization is a specific example of the general problem of control in distributed computation. It is compounded by the fact that centers can start up and shut down independently of one another. A centralized approach to synchronization has been used with success in McROSS simulations. In it, one center acts as a synchronizer for an entire simulator network. When a center starts up it connects to the synchronizer and receives a synchronization message from it. Thereafter, to stay in synch with other centers in the network, the center makes use of the real time clock in the computer it runs on. A distributed approach to synchronization which does not require a synchronizing center is under consideration.

Locally unknown names

Names that are well defined within a simulator network as a whole are not necessarily defined at every node in the network. How should references to such names occurring within centers in which they are not defined be handled? For a specific example in which such a reference is reasonable, reconsider the four node network for simulating Boston-New York traffic. A user controlling the simulator for Boston Terminal who is manually vectoring an aircraft leaving Logan airport might reasonably issue the clearance

fly (V205, PAWLING)

which specifies that the aircraft is to follow Victor Airway #205 to Pawling. Assume that V205 is defined within the geography modules for BOSTRM, BOSCEN and NYCEN and the PAWLING is defined within NYCEN but not within BOSTRM or BOSCEN. The BOSTRM center can't fly the aircraft to Pawling because Pawling is not defined within its airspace. Ideally it should fly the aircraft along V205 to the boundary of the BOSCEN airspace and then hand it off to the BOSCEN simulator. Certainly it should be able to do better than report an error and abort the route procedure. Techniques for handling references to locally unknown names in certain limited contexts are being investigated. However, the general problem of handling such references is an open question.

Program residence

Where should the program (route procedures) required to fly an aircraft through several simulator centers reside? Should the program be associated with the aircraft and passed with it from center to center or should parts of the program be distributed among the relevant centers? The approach currently used in McROSS simulations is to distribute the program and pass only the aircraft and its flight parameters from center to center.

Interruption and resumption of route procedures

Aircraft frequently interrupt their flight plans temporarily in order to avoid severe weather or heavy traffic. The simulation analogy to a flight plan is the route procedure. How should a center handle interruption and subsequent resumption of route procedures? Interrupting the execution of a route procedure in order to follow another one is not difficult. The difficulty arises in determining how to appropriately resume the interrupted procedure. In general, the point of interruption is inappropriate as a resumption point. A considerable amount of (simulated) time can elapse between interruption and resumption during which the flight parameters (position, speed, altitude and heading) of the aircraft can change significantly. Therefore, the usual programming technique of saving the "state" when an interrupt occurs and restoring it after the interrupt has been handled is inadequate. The interruption/resumption problem is made more complex by the possibility that between interruption and resumption the aircraft may fly out of the airspace of one center and into the airspace of another. The current McROSS implementation is not prepared to handle interruption and subsequent resumption of route procedures.

Error handling techniques for distributed systems

The question of how to handle error situations in a distributed computational system is a challenging one. In McROSS considerable attention has been given to making nodes in a simulator network autonomous. The strategy for handling errors is to try to achieve a "local" error recovery whereby a node attempts to preserve its autonomy. As a result, while the actions it

takes are locally optimal in the sense that its continued operation is insured, they may be sub-optimal in the more global context of the entire simulation network.

Errors occurring in inter-node messages are simply handled in the current McROSS implementation. Recall from an earlier section that inter-node messages are submitted to the parsing mechanism of the destination node. When a node receives a message which is syntactically incorrect or semantically meaningless (to it) from a neighbor, it reports the error on its on-line keyboard, sends a message to the neighbor causing the error to be reported on the neighbor's on-line keyboard, and ignores the message. This procedure is locally satisfactory in the sense that it guarantees that messages which are not well-formed cannot cause the node to malfunction. However, if the incorrect message from the neighbor is one critical to the outcome of the simulation, this procedure is not globally acceptable. Ideally, upon detecting an error in a message, the node should engage in a dialogue with its neighbor in an attempt to resolve the error. The difficulty in implementing this strategy is that it is frequently unclear what should be done to resolve an error. Often the cause has been masked by the time the error is detected.

While the simple techniques used in McROSS for error handling have proven adequate, it is clear that more effective techniques can be developed.

CONCLUDING REMARKS

The results of the work reported in this paper are applicable in two areas.

One area is research in air traffic control. Researchers can use the McROSS system to conduct simulation studies preliminary to the design of an automated multi-component air traffic control system. For example, McROSS could be used to evaluate various ways of partitioning responsibility among components of such a system. Or, it could be used to compare different strategies for automated scheduling and control of aircraft. Because it exhibits autonomy of parts and the ability to dynamically reconfigure, it could be used to experimentally study the behavior of failure handling techniques for a multi-center system under various air traffic loads.

The other area is the design and implementation of distributed computation systems. The results in this area are incomplete and tentative consisting primarily of insights and attitudes developed from the experience of building and using a distributed computation system. These are summarized by reviewing the goals for the McROSS system in terms of problems they posed and techniques useful in realizing them.

Of the five goals, autonomy of parts and deferral of process/processor binding were the most significant in terms of effort required to achieve them and influence on the appearance of the system to users. Given their realization, the other three goals (the ability to dynamically reconfigure a simulator network, decentralization of control and ports for monitoring) were relatively easy to achieve.

The strategy of implementing parts of the distributed computation by process groups rather than solitary processes contributed significantly to achieving autonomy of parts. The multi-process implementation made it possible to dedicate certain processes at a node to maintaining an interface with other nodes and to dedicate other processes to performing functions crucial to the node's existence. In addition to insulating vital internal node functions from the actions of other nodes, the functional modularity resulting from multi-process nodes had the effect of reducing the complexity of the implementation: each individual process being considerably less complex than an entire node. The multi-process capability which the TENEX operating system supports for each user job was invaluable in carrying out the multi-process strategy. It is unfortunate that few operating systems allow users to create and maintain process structures.

Equally useful in realizing autonomy was the establishment of and strict adherence to protocols for part-part interactions. A center can expect monitors and adjacent centers which are functioning properly to observe protocol and can therefore interpret a breach of protocol as a warning that the offender may be malfunctioning. A consequence of the multi-process implementation of nodes is that interprocess communication occurs within McROSS at two levels: inter-node and intra-node. Use of a protocol for intra-node interactions helps insure that internal node data bases always accurately reflect the condition of the node's interface with other nodes. A useful implementation rule was to reject any technique whose success depended upon the order in which events in different centers or in different processes within the same center occur.

The major problem in implementing deferred process/processor binding was providing a way for parts of the computation to determine the location of logically adjacent parts at run time. The solution used in the current McROSS implementation, which requires run time interaction with the user, is not totally satisfactory. A more satisfactory alternative might be for each part to engage in a network-wide search for logically adjacent parts.

We expect to see a trend toward distributed multi-computer systems in the future. By its existence McROSS demonstrates that the construction of such systems is currently feasible.

ACKNOWLEDGMENT

The McROSS system grew from an idea proposed by Dr. W. R. Sutherland of BBN; we gratefully acknowledge his guidance and enthusiasm.

The realization of the current McROSS implementation is the result of building upon the work of others, most significantly the designers and implementers of the ARPA computer network and of the TENEX operating system.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
Proceedings of AFIPS SJCC 1970

2 W R SUTHERLAND T H MYER E L THOMAS D A HENDERSON
A route oriented simulation system for air traffic control studies
Proceedings of the Fifth Conference on Applications of Simulation December 8-10 1971

3 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA network
Proceeding of AFIPS SJCC 1970

4 S CARR S CROCKER V CERF
HOST-HOST protocol in the ARPA computer network
Proceedings of AFIPS SJCC 1970

5 D G BOBROW J D BURCHFIEL D L MURPHY R S TOMLINSON
TENEX, A paged time sharing system for the PDP-10
Paper presented at the Third ACM Symposium on Operating System Principles October 18-20 1971
published in Communications of the ACM March 1972

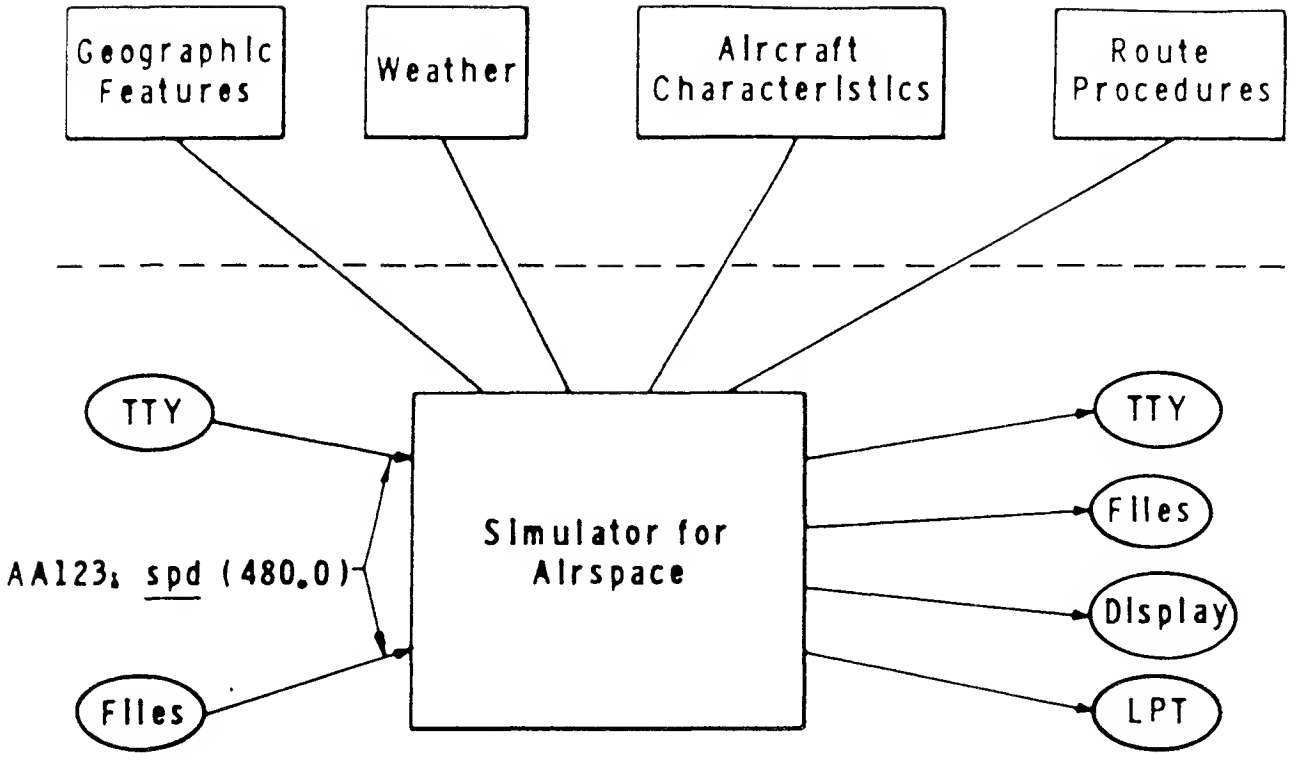
6 *ARPA network current network protocols*
August 1971 Available from the Network Information Center as NIC #7104 at Stanford Research Institute Menlo Park California 94025

7 W NAYLOR J WONG C KLINE J POSTEL
Regarding proffered official ICP
Unpublished memo available from Network Information Center NIC #6728, 1971

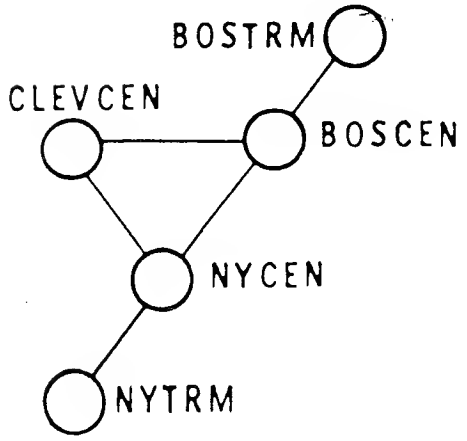
8 A SHOSHANI
A solution to the race condition in the ICP
Unpublished memo available from the Network Information Center NIC #6772 1971

9 E W DIJKSTRA
Cooperating sequential processes
Appears in Programming Languages edited by F Genuys Academic Press New York 1968. Also published as Report EWD123 Dept of Mathematics Technological University Eindhoven The Netherlands 1965

10 D STERN
Weather data
Unpublished memo available from Network Information Center NIC #7692 1971

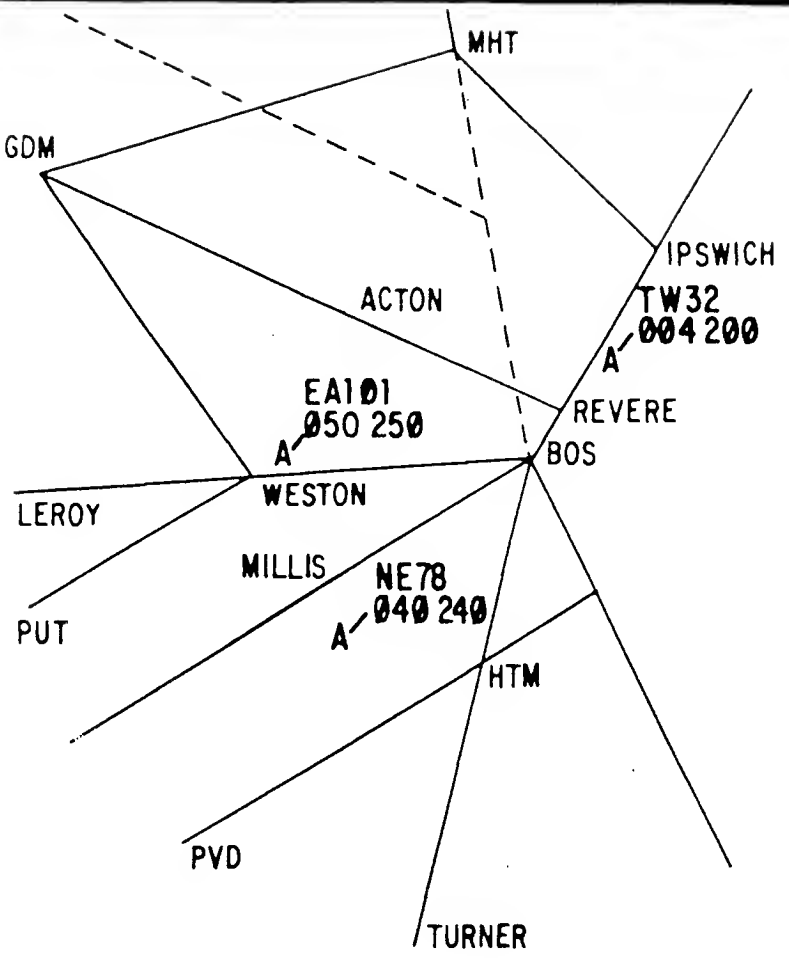


Description of McROSS Simulator, Network:



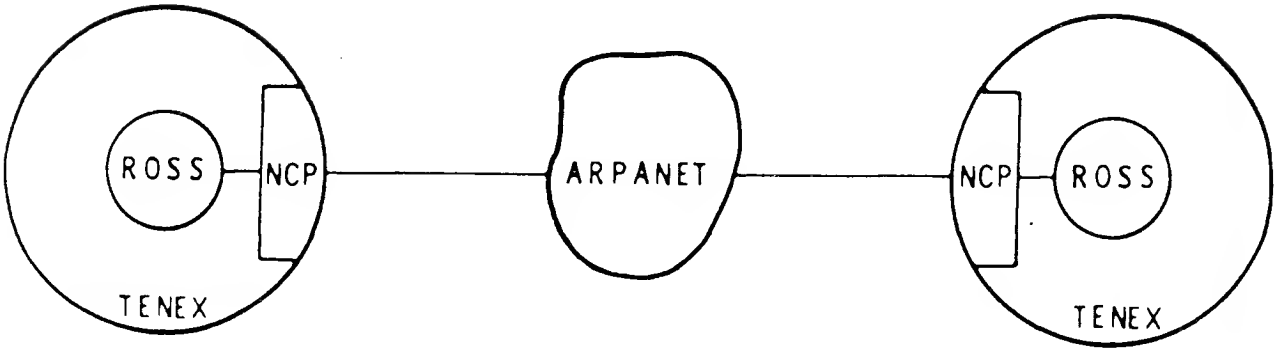

```
netbegin
    netcen  NYTRM, BOSTRM, NYCEN
            BOSCEN, CLEVCEN;
    netcon  (NYTRM, NYCEN);
    netcon  (NYCEN, BOSCEN);
    netcon  (CLEVCEN, NYCEN);
    netcon  (CLEVCEN, BOSCEN);
    netcon  (BOSTRM, BOSCEN);
netend
```

User Defined Airspace for Boston Terminal Area



Components of McROSS System

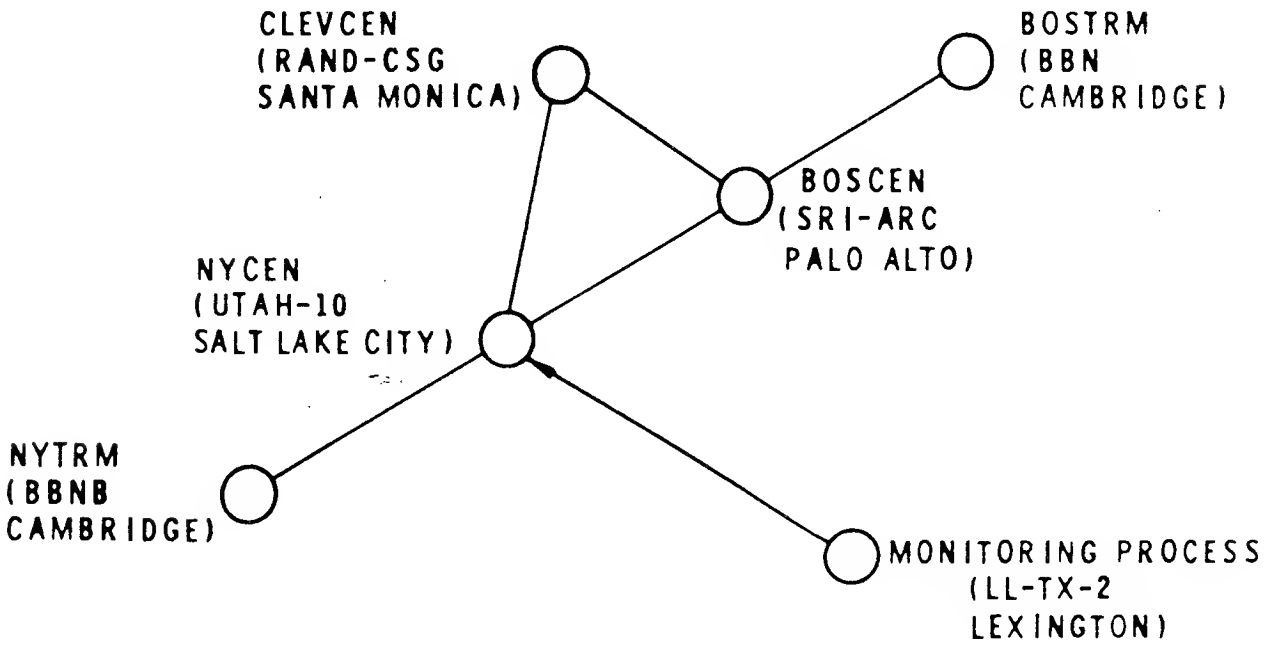
- ROSS
- TENEX Operating System
- ARPANET



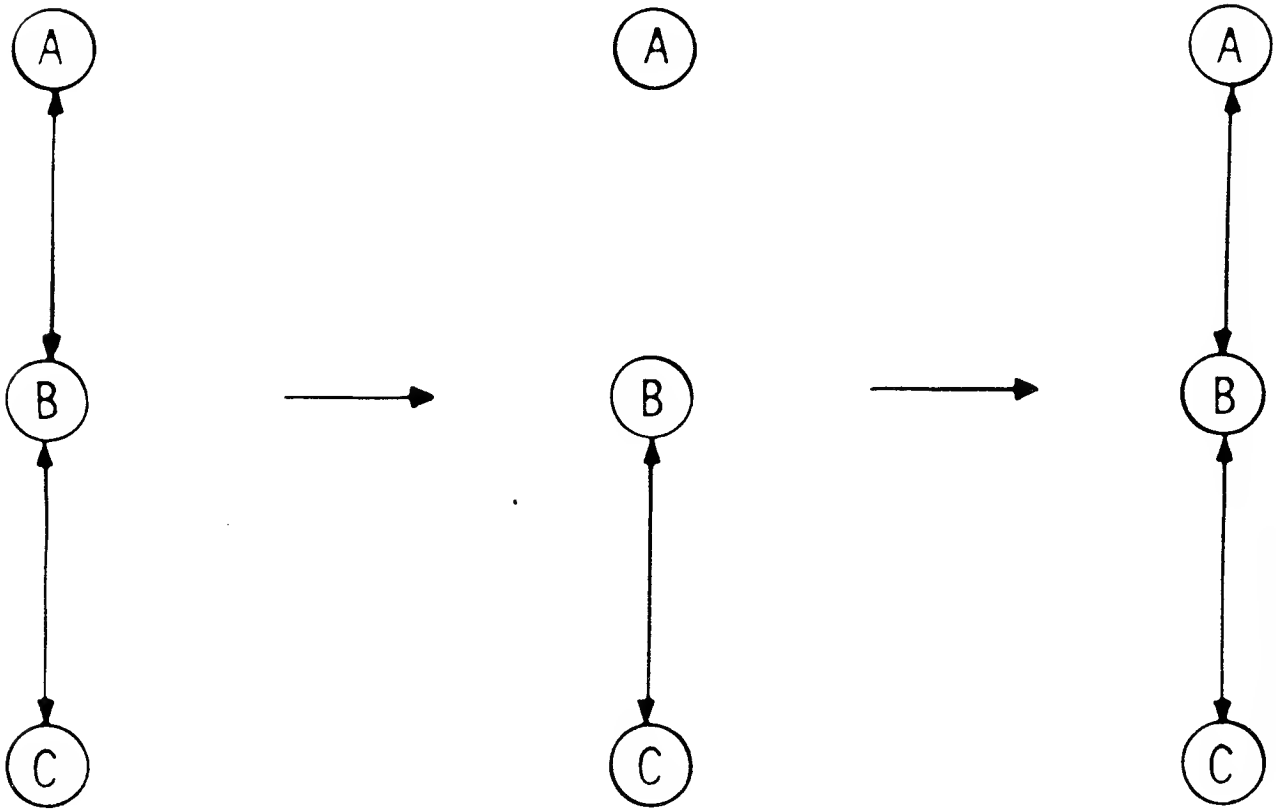
BOSCEN and BOSTRM Establish Connection by:

(at BOSCEN)	(at BOSTRM)
Init (BOSTRM, BBN, SMITH);	Init (BOSCEN, UTAH-10, JONES);
conn (BOSTRM);	conn (BOSCEN);

Configuration During a Typical McROSS Simulation Run



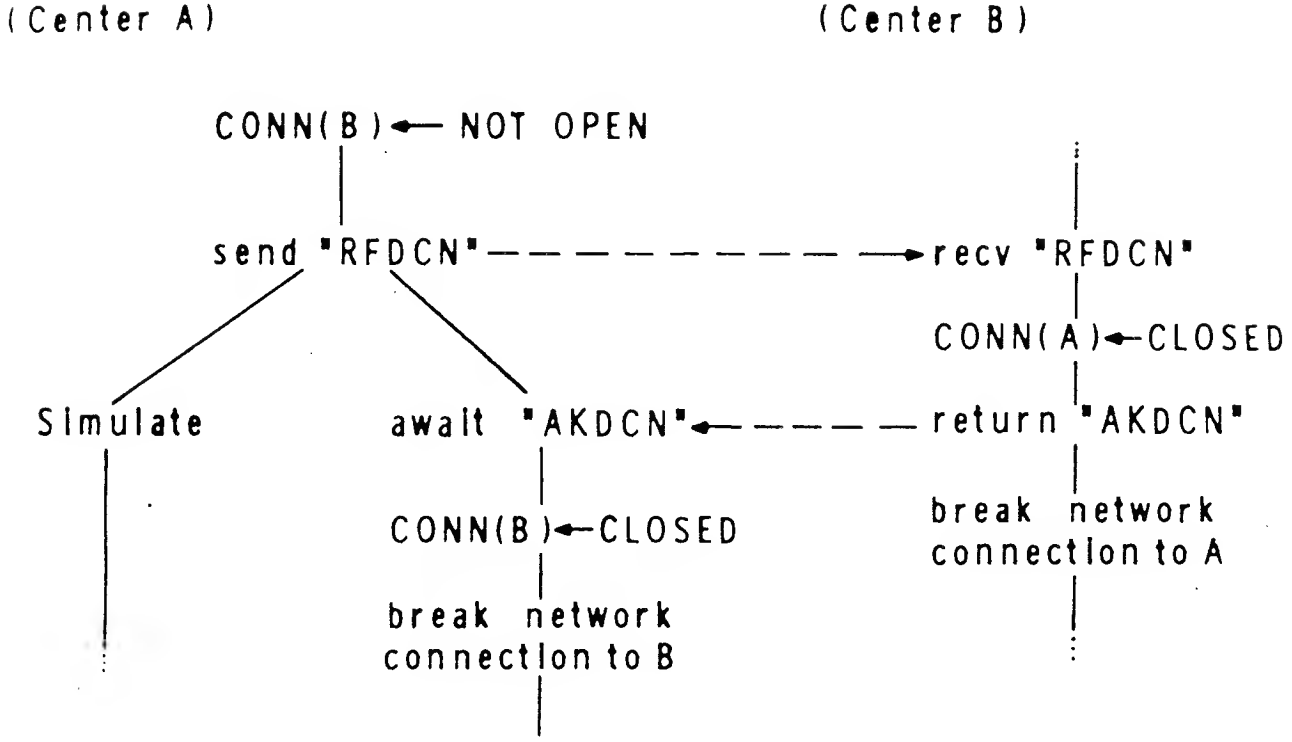
Breaking and Re-establishing Connections



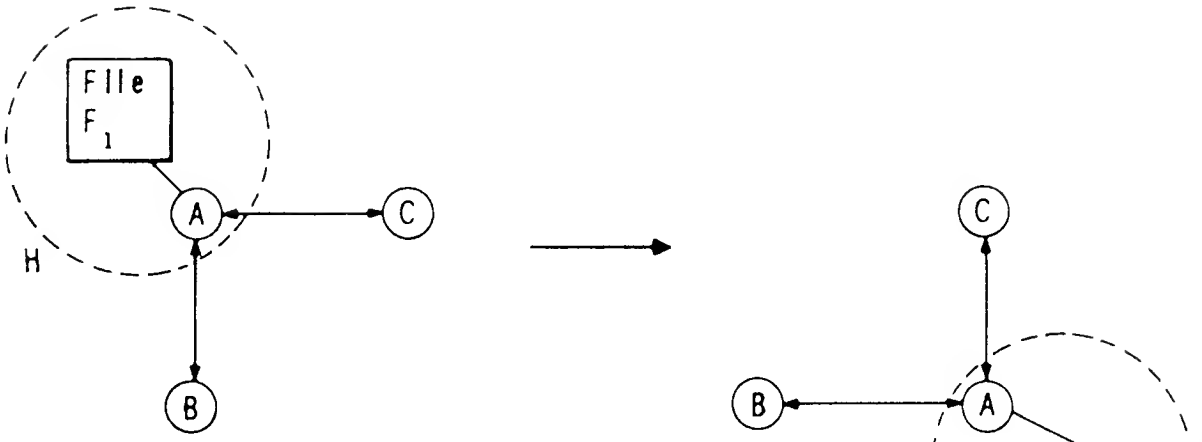
A performs:
dsconn (B)

A performs:
conn (B)

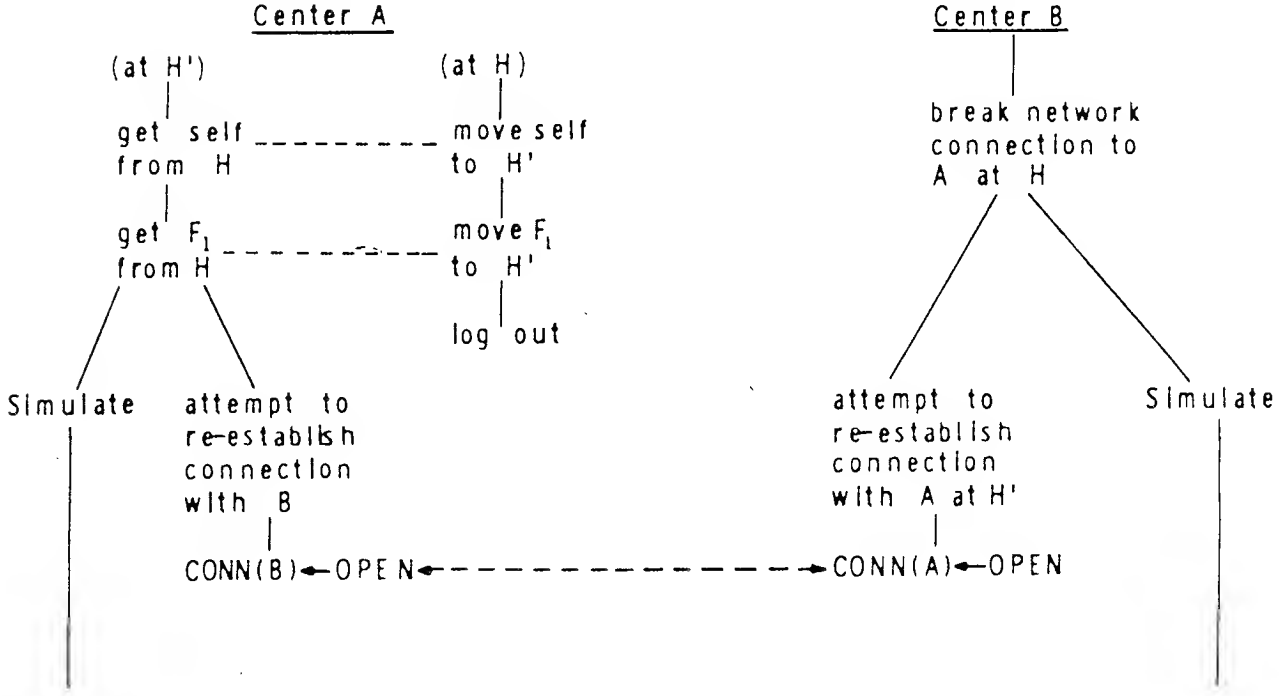
Actions following dsconn (B) by Center A



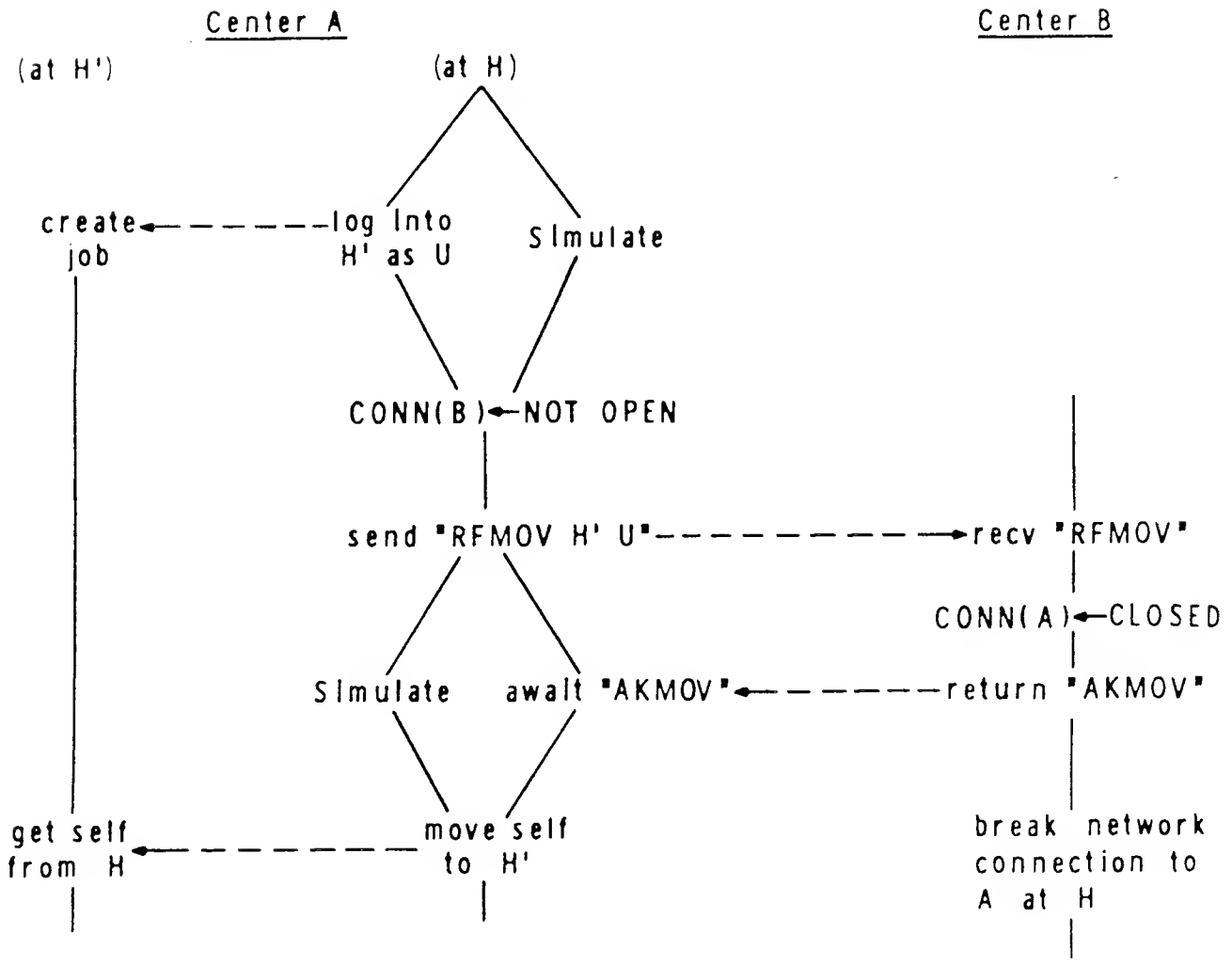
Dynamic Reconfiguration by Center A



A performs:
move (H', U)



Actions following move (H', U) by Center A



Extensions of packet communication technology to a hand held personal terminal

by LAWRENCE G. ROBERTS

Advanced Research Projects Agency
Arlington, Virginia

INTRODUCTION

Electronic communications technology has developed historically almost completely within what might be called the circuit switching domain. Not until the last decade has the other basic mode of communication, packet switching, become competitive. Thus, as a technology, packet communication has only begun to be explored. Circuit switching can be defined in the broad sense as the technique of establishing a complete path between two parties for as long as they wish to communicate, whereas packet switching is where the communication is broken up into small messages or packets, attaching to each packet of information its source and destination and sending each of these packets off independently and asynchronously to find its way to the destination. In circuit switching all conflicts and allocations of resources must be made before the circuit can be established thereby permitting the traffic to flow with no conflicts. In packet switching there is no dedication of resources and conflict resolution occurs during the actual flow perhaps resulting in somewhat uneven delays being encountered by the traffic. Clearly, without the speed and capability of modern computers, circuit switching represented a cheaper and more effective way to handle communications. For radio frequency assignment and telephone exchanges the resource allocation decisions could be made infrequently enough that manual techniques were originally sufficient. Also, since voice was the main information being communicated, the traffic statistics were sufficiently compatible with this approach to make it quite economic for the period. Packet switching of a kind, the telegram, persisted throughout this period but due to the high cost of switching and the limited demand for fast message traffic never attracted much attention.

For almost a century circuit switching dominated the communications field and thus dominated the development of communications theory and technology. Now, within the last decade or less, the advances in digital computers and electronics have, in many cases, reversed the economic balance between circuit and packet communication technology. Perhaps the best proof of this is the economy of the ARPA Network[1–6] for country-wide computer to computer communication, but many other examples are beginning to appear such as the University of Hawaii's ALOHA System[7] utilizing packet radio transmission for console communications and the experiments with digital loops for local distribution. However, most of the experiments with packet communications have been undertaken by computer scientists, and it is not even generally recognized yet in the communications field that a revolution is taking place. Even where the knowledge of one of these experiments has penetrated the communications field, it is generally written off as a possibly useful new twist in communications utilization, and not recognized as a very different technology requiring a whole new body of theory. Throughout the development of the ARPA Network, communication engineers compared it with conventional circuit switched systems but, perhaps unconsciously, used rules of thumb, statistics and experience applicable only to circuit switched systems as a basis for comparison. A century of experience and tradition is not easy to ignore and in fact should not be ignored, only it should be properly classified and segregated as resulting from a different technology.

Packet communication technology is only beginning to be explored but already it is clear that the design of all forms of communications channels and systems should be rethought. As an example of the kind of difference packet communications can make in a perhaps unexpected area, the design of a personal terminal will be explored in some detail. Although such a terminal has never been built, it is most likely completely feasible to build and would provide many unique advantages.

HAND HELD PERSONAL TERMINAL

Let us start with the goal of providing each individual with a pocket-sized, highly reliable and secure communications device which would permit him to send and receive messages to other individuals or to coputers. Leaving the consideration of design alternatives muntil the end, a device fulfilling these objectives is as follows:

Output

Text or graphics displayed on a 2.8″×1″ plasma panel with 80 dots per inch resolution. The screen, divided into 7×10 dot rectangles, using 5×7 characters would hold 8 lines of 32 characters each for a total of 256 characters. Text this size is almost the same size as typewriter print, except that the lines are closer together. The plasma panel has internal storage and is digitally addressed to write or erase a point.

Input

Five capacity or stress sensitive buttons used by the five fingers of one hand simultaneously to indicate one of 31 characters. This five finger keyboard technique was developed by Doug Englebart at SRI[8] to permit users to type with only one hand while working on a display console. Recently the keyboard has become fairly widely used at SRI due to its great convenience. Training time for a new user is evidently less than a day and speeds of 30 words per minute can be achieved.[9] Although somewhat slower than a good typist (½ speed) the speed is clearly sufficient for a terminal device even at 10 words/minute.

Transmission

Each input character will be transmitted to a central controller station using the random access radio transmission techniques developed at the University of Hawaii.[7] The 5 bit character is embodied in a 64 bit packet containing:

30 bits—Terminal Identification Number
8 bits—Character plus alternation bit, or Count
2 bits—Type of packet (CHAR, ACK, CNT, ERR, ST ERR)
24 bits—Cyclic Sum Check
64 bits

All terminals transmit their packets independently and asynchronously on a single frequency and the receiver at the central controller merely listens for a complete packet which has a correct sum check. If two terminals' transmissions overlap the sum check will be wrong, and the terminals will retransmit when they find they don't receive an acknowledgment. Retransmission time-out intervals are randomized between the terminals to avoid recurrence of the problem. Upon receipt of a good packet, the central station transmits a display-acknowledgment packet back to the terminal on a second frequency. This 144 bit packet contains a 70 bit display raster field and an 8 bit position on the screen. The display raster is a 7×10 dot array for the character sent in and the position includes 3 bits for vertical by 5 bits for horizontal. Current position information for each active user is kept by the central station by user ID in a hash table. Thus, the individual terminal needs no character generation logic, position advancement logic, or any other local verification of the input since the response from the central station both acknowledges the character and displays it in an input text line at the top of the display. If a character display-acknowledgment is somehow lost in transmission the terminal will continue to time-out and retransmit the character. The central station must somehow differentiate this from a new character. This is achieved by an alternation bit[10,11] in the terminal's packet which is complemented for each *new* character. On a repeat the bit is the same as previously and the central station just retransmits the same character and position again. When a prearranged terminating character is sent the central station examines the message and takes an appropriate action. Considerable flexibility exists here, and operational modes could be established. However, the first message of a sequence should contain a destination as the first item. This might be the ID of another terminal in the same area, it might be the address of a service computer or it might be the ID of another terminal halfway around the world. In the latter two cases, a more global network such as the ARPA Network comes into play. It would be perfectly feasible for a message to another terminal to be sent to a central or area-coded directory computer to locate the particular control station the other terminal was near. Note that the location of neither man was given to the other, only the message and the ID of the sender. (Based on ARPA Network cost estimates and international satellite tariff trends, such a message exchange should cost less than 0.1 cents, independent of distance.)

Reception

At any time when a message destined for a terminal arrives at the central control station, a transmission to

the terminal may begin, character by character, each in its own 144 bit packet as follows:

30 bits—Terminal Identification Number
70 bits—7×10 dot pattern, character display
8 bits—position of character
1 bit —alternation bit
1 bit —broadcast mode
3 bits—Message Type (Response, initial, normal)
8 bits—Characters Left in message
24 bits—Cyclic Sum Check
144 bits

The terminal must always be checking all transmission to detect those with its ID and a correct sum check. When one occurs which is not a "response" to an input character, a message is being sent. The first character of a message is marked type "initial," and has the count of the remaining characters. Each character is displayed where the central station placed it. Following the "initial" character "normal" characters are checked to make sure the count only decreases by one each time. After the character with count zero, an acknowledgment type packet is sent by the terminal. If this is lost (as it may be due to conflicts) the central control will retransmit the final character over again without complementing the alternation bit until it is acknowledged (or it determines the station is dead). If a count is skipped the terminal sends a CNT ERR message with the count of the character expected. The transmitter then starts over at that count. If a "normal" type character is received before an "initial" type a ST ERR message is sent and the message is restarted. A broadcast bit is included which overrides the ID check for general messages.

Security

Since all transmissions are digital, encryption is possible and would be important no matter what the application, military or civilian. Most private uses such as personal name files, income-expense records, family conversations, etc., would be far more sensitive than current computer console use.

Bandwidth

Personal terminals for occasional use for message exchange, maintaining personal files, querying computer data bases for reference data, etc., would not lead to very heavy use, probably no more than two query-responses per hour. The query we might estimate at 64 characters in length and the response at 256. (Clearly 256 character response could also consist of an 80×224 point graphic display since each character is sent as a full 7×10 raster.) The average bandwidth consumed by each terminal is therefore 2.3 bits/second transmitted and 25.6 bits/second received. The random access technique used for transmission requires the channel bandwidth to be six times the average bandwidth actually utilized in order to resolve all conflicts properly. Thus, the terminal transmission bandwidth consumption is 14 bits/second, still less than the receiver bandwidth needed. Thus, the central control station's transmitter bandwidth is the limiting factor assuming equal bandwidths on both transmitter and receiver. If a 50KHz bandwidth is used for each and modulated at 50K bits/sec, then a total of 2000 terminals can be accommodated. Of course this number depends on the activity factor. At one interaction every two minutes a data rate equal to average time shared console use is obtained and even at this activity 130 terminals can be supported, more than most time-sharing systems can support. With 50 KB channels, the time required to write 256 characters is about one second. Lower bandwidths require increased time, thus, 10KB (5 sec write time) would be the lowest bandwidth reasonable. Even at this bandwidth, with the estimated 2 interactions per hour, 400 terminals could be supported.

COMPARISON

Comparing the effect of the packet technology with the same terminal operating with preassigned Frequency or Time Division Multiplexed channels (ignoring the losses due to TDM sync bits or FDM guard bands) the circuit oriented terminal would require a 40 bit/sec transmit channel and a 4KB receive channel if a 5 sec write time is to be achieved. For 400 terminals with a 5 sec write time, the circuit method would require a total of 1.6 Megabits/sec bandwidth whereas the packet method only requires 20 Kilobits/sec bandwidth. Thus, the circuit technology requires a factor of 80 more bandwidth than the packet technique. Of course, the circuit mode terminals could interact more often within the same bandwidth right up to continual rewrite of the display every five sec, but you would have to massively reshape the user statistics to suit the technology.

Another possibility, to design the terminal so that it performed more effectively in a circuit oriented mode, would be to put character generation logic and position logic in the terminal. This would considerably increase the cost of the terminal, which originally had very little logic except shift registers. The result of adding this logic, however, is to reduce the bandwidth by a factor

of 10 to .16MB or still 8 times the packet technique. The same logic would help reduce the packet size but, in order to maintain the graphic output capability and gross simplicity, it does not seem to pay.

CONCLUSION

As can be seen from the example, packet technology is far superior to circuit technology, even on the simplest radio transmission level, so long as the ratio of peak bandwidth to average bandwidth is large. Most likely, the only feasible way to design a useful and economically attractive personal terminal is through some type of packet communication technology. Otherwise one is restricted to uselessly small numbers of terminals on one channel. This result may also apply to many other important developments, only to be discovered as the technology of packet communication is further developed.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
SJCC 1970

2 F E HEART R E KAHN S M ORNSTEIN
W R CROWTHER D C WALDEN
The interface message processor for the ARPA network
SJCC 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
SJCC 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
SJCC 1970

5 S CARR S CROCKER V CERF
HOST-HOST communication protocol in the ARPA network
SJCC 1970

6 L G ROBERTS
A forward look
Signal Vol XXV No 12 pp 77-81 August 1971

7 N ABRAMSON
THE ALOHA System—Another alternative for computer communications
AFIPS Conference Proceedings Vol 37 pp 281-285 November 1970

8 D C ENGELBART W K ENGLISH
A research center for augmenting human intellect
AFIPS Conference Proceedings Vol 33 p 397 1968

9 D C ENGELBART
Stanford Research Institute Menlo Park Calif (Personal communication)

10 W C LYNCH
Reliable full-duplex file transmission over half-duplex telephone lines
Communications of the ACM Vol 11 No 6 pp 407-410 June 1968

11 K A BARTLETT R A SCANTLEBURY
P T WILKINSON
A note on reliable full-duplex transmission over half-duplex links
Communications of the ACM Vol 12 No 5 pp 260-261 May 1969

COMMUNICATIONS TECHNOLOGY

CHANNELS

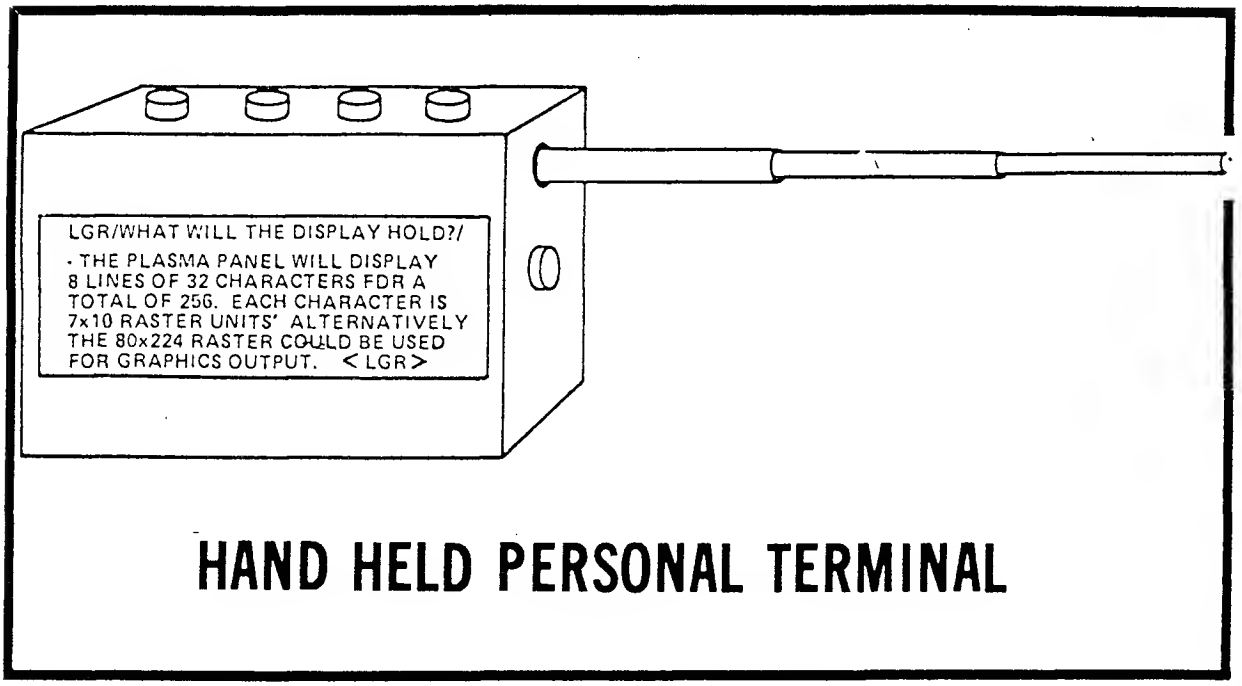
- END-TO-END CIRCUIT
- INFREQUENT DECISIONS

  - TELEPHONE (1876)
  - RADIO (1901)
  - FACSIMILE (1924)

PACKETS

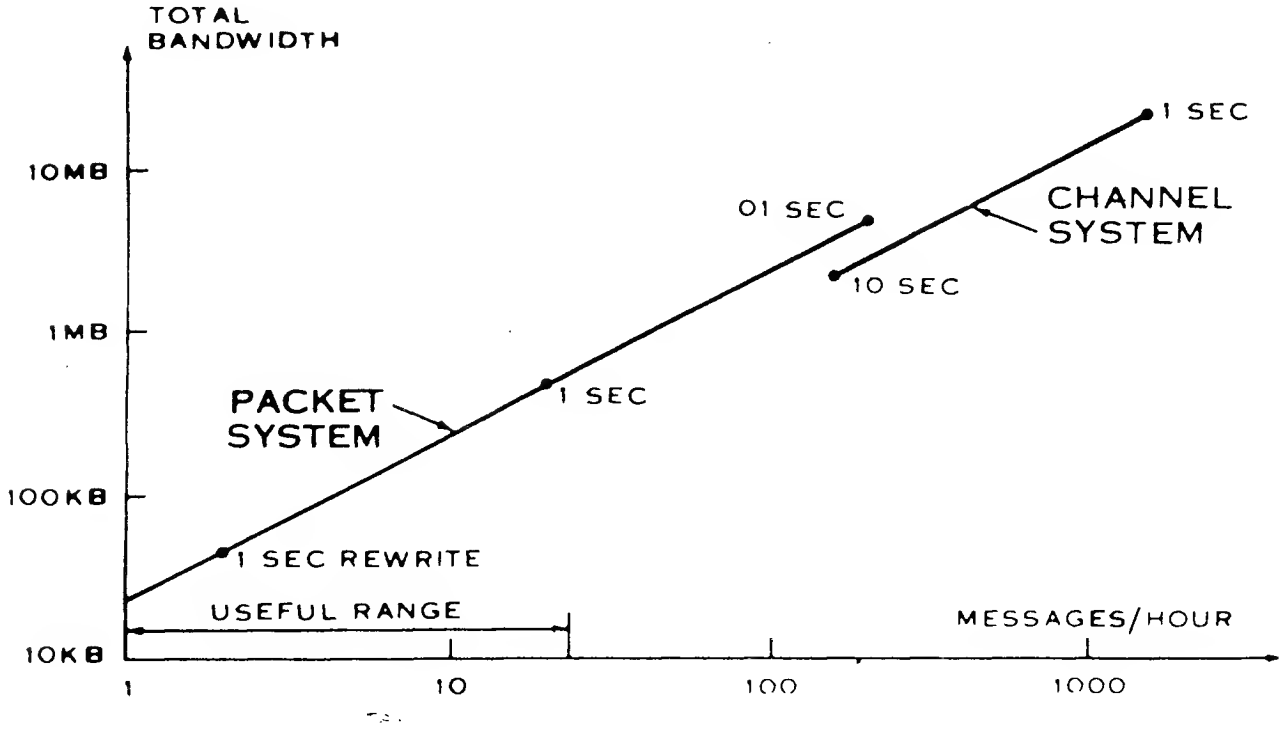
- ADDRESSED MESSAGE
- DYNAMIC RESOURCE ALLOCATION

  - TELEGRAPH (1844) AND DESCENDANTS (1960'S)
  - ARPANET (1969)
  - ALOHA NET (1969)

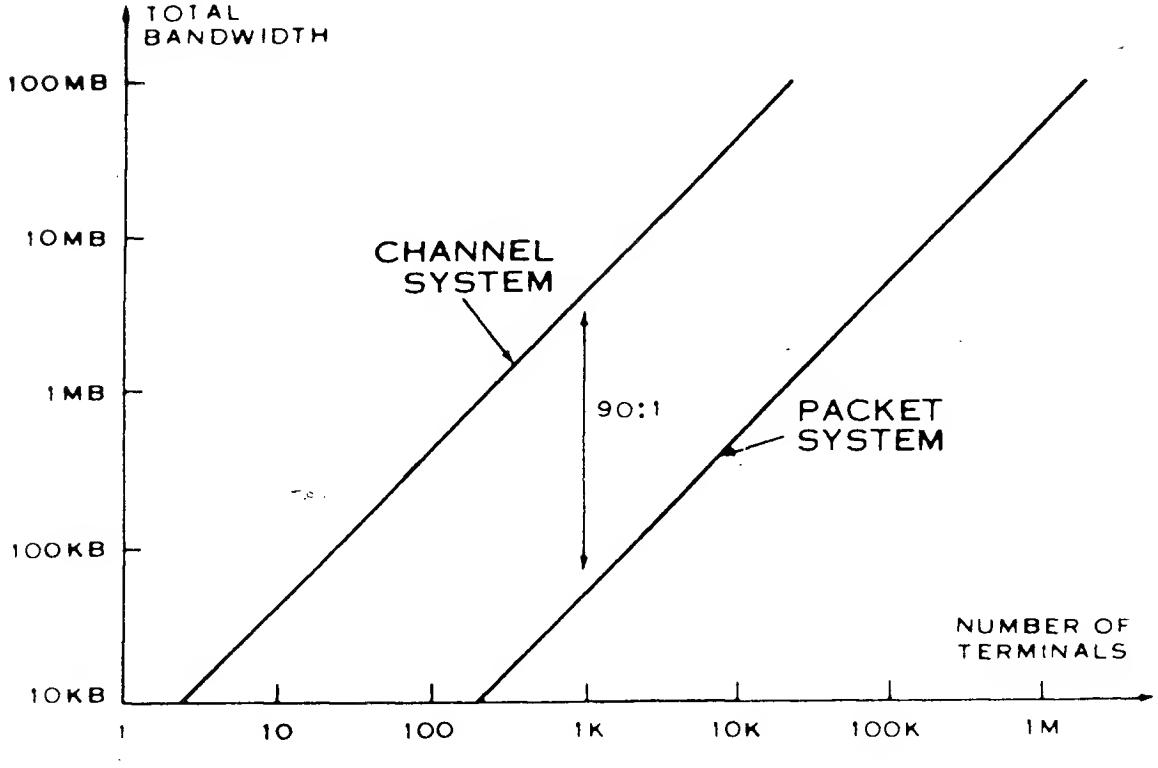


HAND HELD PERSONAL TERMINAL

BANDWIDTH REQUIREMENTS FOR 1000 TERMINALS



PERFORMANCE WITH ≤ 5 SEC DISPLAY REWRITE FOR A RATE OF 2 MESSAGES/HOUR



THE ARPA NETWORK

☆ ☆ ☆ ☆ ☆

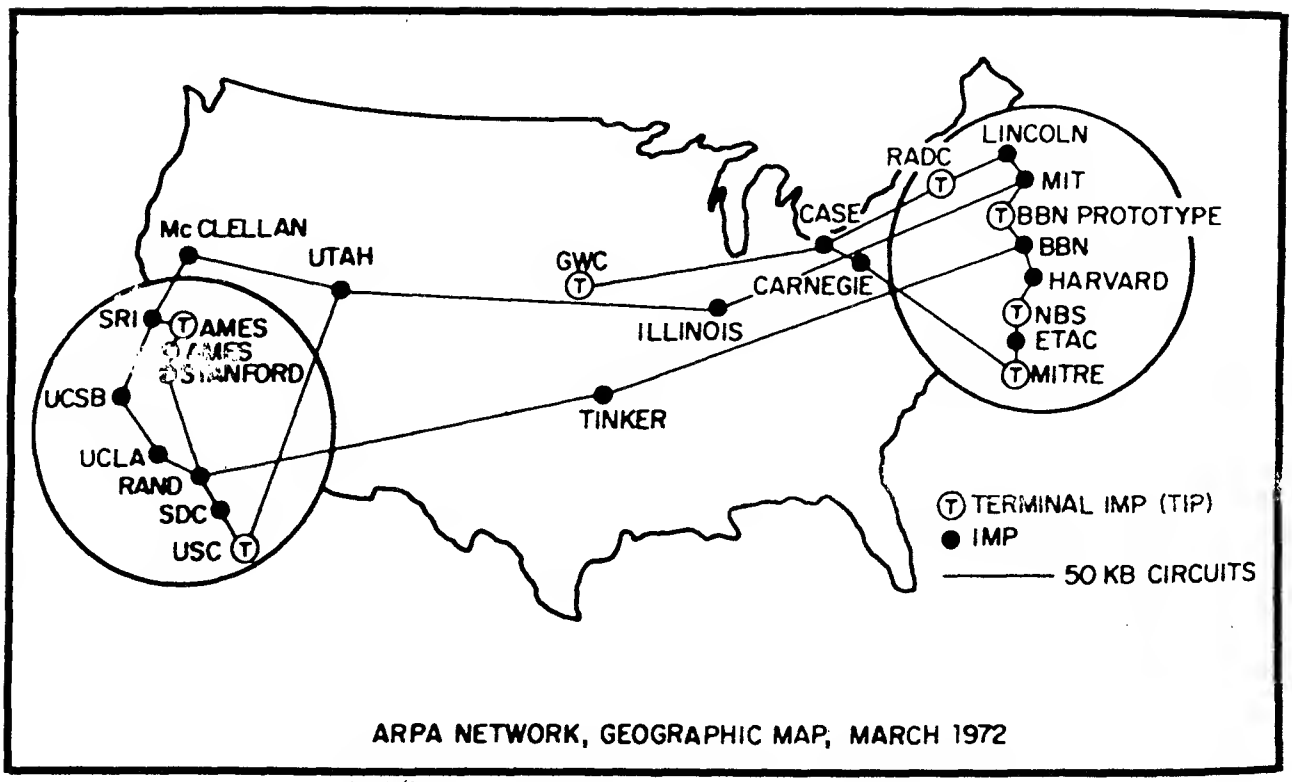
A session presented at the Spring Joint Computer Conference, Atlantic City, May 16, 1972 and chaired by S. D. Crocker, Advanced Research Projects Agency

- The Terminal IMP for the ARPA Computer Network
- Computer Communication Network Design - Experience with Theory and Practice
- Function-Oriented Protocols for the ARPA Computer Network
- McROSS - A Multi-Computer Programming System
- Extension of Packet Communication Technology to a Hand-Held Personal Terminal

ARPA ADVANCED RESEARCH PROJECTS AGENCY

WASHINGTON, D. C.

IN THIS BOOKLET, THE SESSION CHAIRMAN HAS APPENDED TO SOME OF THE PAPERS A SUBSET OF THE SLIDES PRESENTED AT THE SJCC SESSION.



ARPA NETWORK, GEOGRAPHIC MAP, MARCH 1972

The Terminal IMP for the ARPA computer network*

by S. M. ORNSTEIN, F. E. HEART, W. R. CROWTHER, H. K. RISING, S. B. RUSSELL and A. MICHEL

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

A little over three years ago the Advanced Research Projects Agency of the Department of Defense (ARPA) began implementation of an entirely new venture in computer communications: a network that would allow for the interconnection, via common-carrier circuits, of dissimilar computers at widely separated, ARPA-sponsored research centers. This network, which has come to be known as the ARPA Network, presently includes approximately 20 nodes and is steadily growing. Major goals of the network are (1) to permit resource sharing, whereby persons and programs at one research center may access data and interactively use programs that exist and run in other computers of the network, (2) to develop highly reliable and economic digital communications, and (3) to permit broad access to unique and powerful facilities which may be economically feasible only when widely shared.

The ARPA Network is a new kind of digital communication system employing wideband leased lines and message switching, wherein a path is not established in advance and instead each message carries an address. Messages normally traverse several nodes in going from source to destination, and the network is a store-and-forward system wherein, at each node, a copy of the message is stored until it is safely received at the following node. At each node a small processor (an *Interface Message Processor*, or *IMP*) acts as a nodal switching unit and also interconnects the research computer centers, or *Hosts*, with the high bandwidth leased lines.

A set of papers presented at the 1970 SJCC[1–6] described early work on the ARPA Network in some detail, and acquaintance with this background material (especially Reference 2) is important in understanding the current work. The present paper first discusses major developments that have taken place in the network over the last two years. We then describe the *Terminal IMP*, or *TIP*, a development which permits direct terminal access to the network. Finally we mention some general issues and discuss plans for the next stages in development of the network.

THE DEVELOPING NETWORK

The initial installation of the ARPA Network, in 1969, consisted of four nodes in the western part of the United States. A geographic map of the present ARPA Network is shown in Figure 1. Clearly, the most obvious development has been a substantial *growth*, which has transformed the initial limited experiment into a national assemblage of computer resources and user communities. The network has engendered considerable enthusiasm on the part of the participants, and it is increasingly apparent that the network represents a major new direction in both computer and communications technology.

Figure 2 is a logical map of the network, where the Host computer facilities are shown in ovals, all circuits are 50 kilobits, and dotted circuits/nodes represent planned installations. On this figure certain nodes are listed as a "316 IMP"; this machine is logically nearly identical to the original IMP, but can handle approximately two-thirds of the communication traffic bandwidth at a cost savings of approximately one-half. The original IMP includes a Honeywell 516 computer, and more recently Honeywell began to market the 316 computer as a cheaper, downward-compatible machine. As the network has grown, sites were identified which did not require the full bandwidth of the original IMP, and a decision was made to provide an IMP version built around the 316 computer. Also shown in Figure 2 are certain nodes listed as "TIP"; this new machine

* This work was sponsored by the Advanced Research Projects Agency under Contract No. DAHC15-69-C-0179.

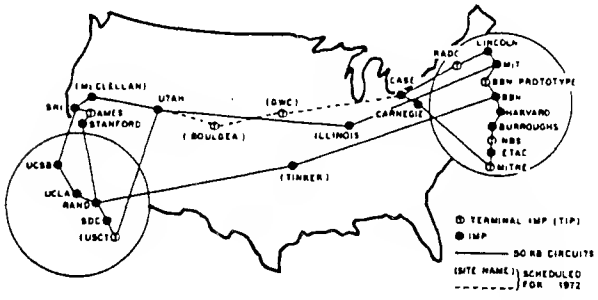


Figure 1—ARPA Network, geographic map, December 1971

is discussed in detail later in this paper. Site abbreviations shown on Figures 1 and 2 are explained in Table I.

As the network has grown, a great deal of work has been concentrated on the development of Host-to-Host protocol procedures. In order for programs within one Host computer system to communicate with programs in other Hosts, agreed-upon procedures and formats must be established throughout the network. This problem has, as predicted, turned out to be a difficult one. Nonetheless protocol procedures have evolved and are being accepted and implemented throughout the net. At the present writing, many of the Hosts have working "network control programs" which implement this protocol. Protocol development is more fully reported in a companion paper,[6] but we wish to make a general observation on this subject: the growth of the network has dynamically catalyzed an area of computer science which has to do with the quite general problem of *how programs should intercommunicate*, whether in a single computer or between computers. Thus the evolution of the Host-to-Host protocol represents a side benefit of the network that reaches well beyond its utility to the network alone.

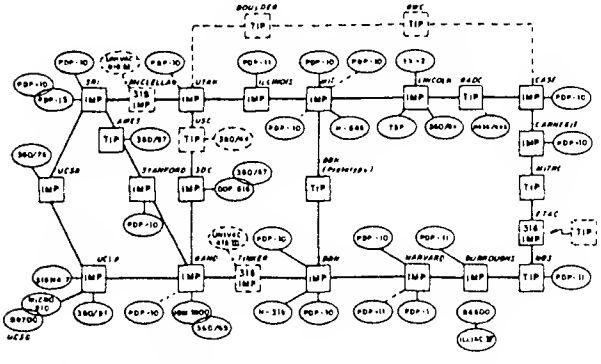


Figure 2—ARPA Network, logical map, December 1971

Since both hardware and software network connections must be implemented by each Host, it is important that the external characteristics of the IMP be relatively stable. This stability has been carefully maintained, while at the same time internal operation of the IMP program has undergone extensive revision and improvement. For example, trouble reporting, statistics gathering, and test procedures have been substantially improved. In addition to improvements that have already been incorporated into the program, there have also been extensive studies of performance and message flow control.[7] These studies have pointed up areas of vulnerability to perverse

TABLE I—Site Abbreviations

NCAR	National Center for Atmospheric Researeh.
GWC	Global Weather Central
SRI	Stanford Research Institute
MC CLELLAN	McClellan Air Force Base
UTAH	University of Utah
ILLINOIS	University of Illinois
MIT	Massachusetts Institute of Technology
LINCOLN	M.I.T. Lincoln Laboratory
RADC	Rome Air Development Center
CASE	Case Western Reserve
AMES	N.A.S.A. Ames Research Center
USC	University of Southern California
UCSB	University of California at Santa Barbara
STANFORD	Stanford University
SDC	Systems Development Corporation
BBN	Bolt Beranek and Newman
CARNEGIE	Carnegie University
MITRE	MITRE Corporation
ETAC	Environmental Technical Applications Center
UCLA	University of California at Los Angeles
RAND	Rand Corporation
TINKER	Tinker Air Force Base
HARVARD	Harvard University
BURROUGHS	Burroughs Corporation
NBS	National Bureau of Standards

heavy traffic patterns and have suggested still other possible improvements in the routing and flow control mechanisms. Potential changes are presently being being studied in some detail and will be incorporated into one or more forthcoming major revisions of the program. They will hopefully anticipate problems which might be expected to arise as traffic flow in the network becomes heavier.

Somewhat belatedly in the network design, the need to connect a single IMP to several Hosts was recognized. This required multiple Host interfaces at the IMP as well as more complex IMP software. Further, the various Host computers at a site are often physically

distant from one another, thus requiring an increase in the maximum allowable physical separation between a Host and its IMP. To connect to an arbitrarily remote Host would have meant a communications interface capable of attachment to common-carrier circuits via modems. It would furthermore have required cooperative error control from the Host end. At the time, we chose not to modify the logical way in which the IMP dealt with the Host and instead we provided more sophisticated line drivers which would handle distances of up to two thousand feet. Several such "Distant Host" installations are now working in the network. Unfortunately, as the network has grown, new sites have appeared where still greater Host/IMP distances are involved. The present scheme does not include error control, and use of this scheme over greater distances is not appropriate. At the present time we are therefore considering how best to arrange IMP/Host connections over large distances and additional options will be required in this domain.

Another facility which has been tested is the ability of the IMPs to communicate over 230.4 kilobit phone lines instead of 50 kilobit lines. A short, fast link was incorporated into the network for a brief period and no problems were encountered. To date, network loading has not justified upgrading any operational network circuits to 230.4 kilobits, but this will be considered as loading rises.

Substantial effort has gone into traffic and trouble reporting. A Network Control Center (NCC) has been built at Bolt Beranek and Newman Inc. in Cambridge, where a small dedicated Host computer receives reports each minute from every IMP on the network. Traffic summaries and status and trouble reports are then generated from this material. Specifically, a logger records any changes that the IMPs report to the NCC; it records line errors and IMP storage counts when they exceed certain limits, as well as unusual events, such as reloading from the net, checksum errors, etc. Figure 3 shows an example of this log. (The comments, in parentheses to the right, are not part of the log but have been added to explain the meaning of the entries.) In addition to this detailed log of interesting events, one may at any time obtain a quick summary of the status of the network. Finally, detailed and summary logs of Host and line traffic are produced. The NCC is a focal point not only for monitoring the network but also for testing and diagnosing remote troubles. Lines throughout the network can be looped from here in order to isolate difficulties. Personnel of the center coordinate all debugging, maintenance, repair and modification of equipment.

```
1300     JUNE 16 197-     ARPA NETWORK LOG                PAGE 11

1306  IMP     6:  HOST 1 UP                  (Host 1 at MIT came up)
      IMP     1:  SS2 ON                     (Sense switch 2 was thrown at UCLA
1301  IMP     1:  10 SEC STAT ON             (UCLA is using IMP statistics)
1305  IMP     1:  10 SEC STAT OFF            (UCLA has finished)
      IMP     1:  SS2 OFF                    (and turned the switch off)

1307  IMP     4:  UP #####                   (Utah IMP was down, and has come u

      IMP     4:  RELOADED FROM NET          (A neighbor IMP sent Utah a copy of
      IMP     4:  VERSION 2614               (the IMP program over a phone line)
      IMP     4:  HOST 1 UP                  (Host 1 at Utah is now up)

1310  LINE    4:  UP #####                   (one of Utah's lines is up)

      LINE   10:  UP #####                   (another is up)

      LINE   15:  DOWN #####                 (but the third is making errors)

1317  LINE   15:  ERRORS MINUS 13/81         (Utah sees 20% error rate)
      LINE   15:  ERRORS PLUS 7/81           (the other end sees a 10% rate)
      IMP     6:  HOST 1 DN                  (Host 1 at MIT went down)
1320  LINE   15:  ERRORS MINUS 0/81         (the line is error-free)
      LINE   15:  ERRORS PLUS 0/81           (in both directions)

1321  LINE   15:  UP #####                   (the line is declared usable)
```

Figure 3—Typical segment of NCC log

DIRECT TERMINAL ACCESS

During the early phases of network development a typical node has consisted of one or more large time-shared computer systems connected to an IMP. The IMPs at the various sites are connected together into a subnet by 50 kilobit phone lines and the large Host computers communicate with one another through this subnet. This arrangement provides a means for sharing resources between such interconnected centers, each site potentially acting both as a user and as a provider of resources. *This total complex of facilities constitutes a nationwide resource which could be made available to users who have no special facilities of their own to contribute to the resource pool.* Such a user might be at a site either with no Host computer or where the existing computer might not be a terminal-oriented time-sharing system.

A great deal of thought went into considering how best to provide for direct terminal access to the network. One possibility, which would have essentially been a non-solution, was to require a user to dial direct to the appropriate Host. Once connected he could, of course, take advantage of the fact that that Host was tied to other Hosts in the net; however, the network lines would not have been used to facilitate his initial connection, and such an arrangement limits the terminal bandwidth to what may be available on the switched common-carrier networks.

A similar solution was to allow terminals to access the network through a Host at a nearby node. In such a case, for example, a worker in the New England area wishing to use facilities at a California site might connect into a local Boston Host and use that Host

as a tap into the network to get at the facilities in California. This approach would have required Hosts to provide hardware access facilities for many terminals which would use their systems only in passing. For many Hosts, the kinds of terminals which can be connected directly are limited in speed, character set, etc. In terms of reliability, the user would have been dependent on the local Host for access: when that Host was down, his port into the network would be unavailable. Furthermore, the Hosts would have been confronted with all of the problems of composing terminal characters into network messages and vice versa as well as directing these messages to the proper terminals and remote Hosts. Time-sharing systems are generally already overburdened with processing of characters to and from terminals and many are configured with front end processors employed explicitly to off-load this burden from the main processor. Increasing the amount of such work for the Hosts therefore seemed unreasonable and would have resulted in limiting terminal access. Instead, a completely separate means for accessing the network directly seemed called for: an IMP with a flexible terminal handling capability—a *Terminal IMP*, or *TIP*.

One of the fundamental questions that arises when considering this problem is: what is the proper distribution of computational power among the remote big facility, the terminal processor, and the terminal? Shall the terminal processor be clever, have sizable storage, be user programmable, etc., or shall it be a simpler device whose basic job is multiplexing in a flexible way? Serious work with interactive graphics seems to require the terminal to include, or be in propinquity to, a user-programmable processor and considerable storage. To date, such work has primarily been done with terminals attached directly to a Host. Some elaborate terminals, such as the Adage, include a powerful processor. Other kinds of terminals, including Teletype-like devices, alphanumeric displays, or simple graphic displays (excluding serious interactive graphics), do *not* require a user-programmable local processor or significant local storage.

We believe that the great majority of potential groups needing access to the ARPA Network will not need powerful interactive graphics facilities. Further, for the minority that will need powerful terminals, we believe that individual terminals with built-in or accompanying processors will become more common. For these reasons, we decided that the terminal processor should be simple and not programmable from the terminals. *The computational load and the storage should be in the Hosts or in the terminals and not in the terminal processor.* This simple multiplexing approach is amenable to some standardization and is philosophically close to the original IMP notion of a standard nodal device.

Another major question we faced was whether to build a separate terminal handler and then connect it to the IMP or to build an integrated unit that was housed in a common cabinet and used the IMP processor. One advantage of the two-machine approach is that it isolates the IMP functions from the terminal functions, thereby providing a barrier of safety for the net. This approach also provides the processing power of two machines and a potentially greater degree of user freedom in modifying or writing programs. Another interesting reason for considering separate machines was to reduce the large cost associated with I/O equipment (such as line controllers) by making use of the extra processing power. We discussed with several manufacturers the possibility of bringing terminal wires "into" their processors and decoding the basic line information directly in software. However, even with some of the new state-of-the-art machines, like the Meta-4, with fast 90 nsec read-only memories to handle character decoding, the I/O cost was still high, and in large part the necessary I/O equipment was yet to be designed. We therefore concluded that it was still somewhat early to proceed in this fashion, and two processors did not appear to save I/O equipment.

The principal disadvantages of the two-machine approach were the higher initial cost, the difficulties of maintaining two machines, and the software problems of dealing with two machines. In particular, the communication between two machines would require two hardware Host interfaces, and two software Host/IMP programs. This would result in a much poorer communication between the IMP program and the terminal handling program than would exist in a single machine. In either case, one machine or two, the

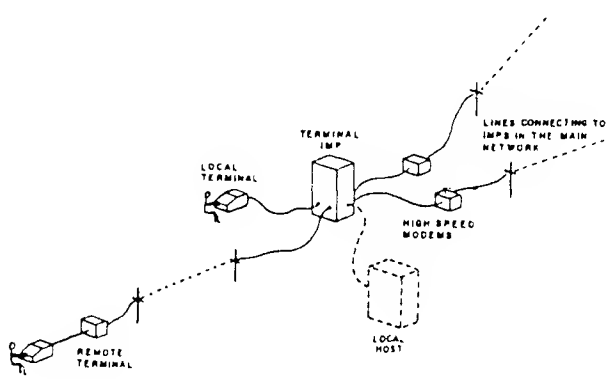


Figure 4—A TIP in the network

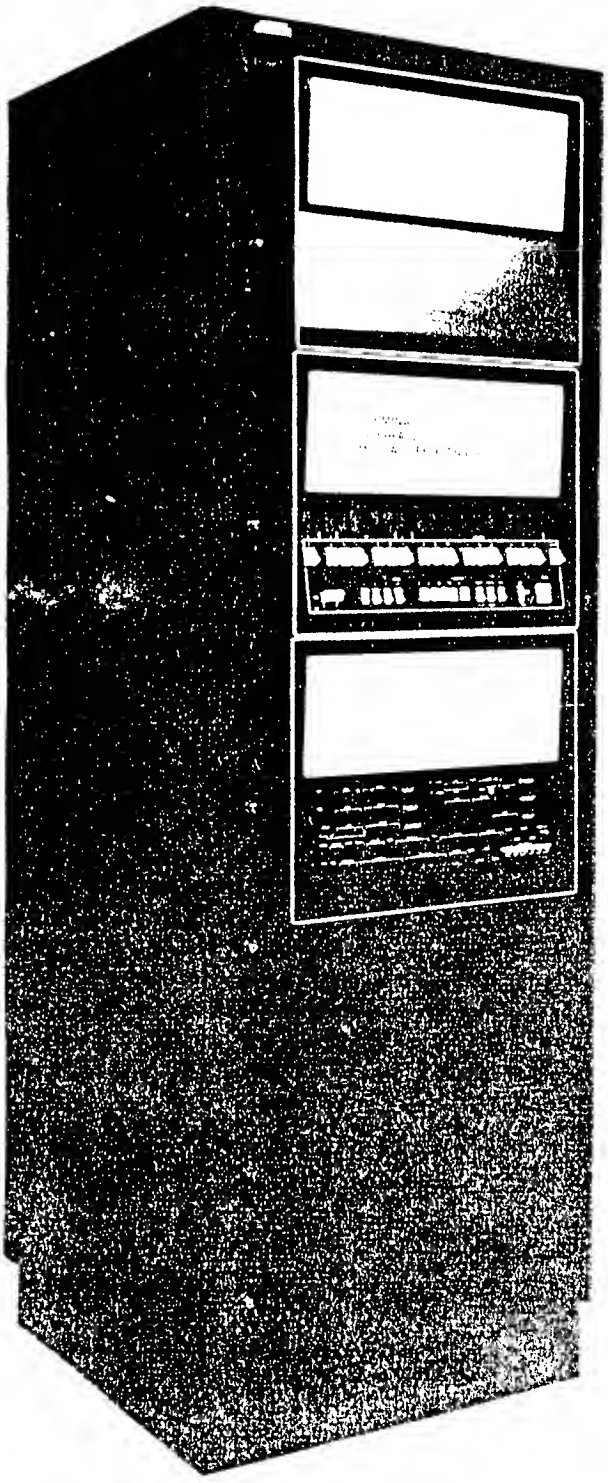


Figure 5—Photograph of TIP

terminal handling process required the implementation of a version of Host protocol.

We finally decided that the least expensive and most sensible technical alternative was to build the IMP and the terminal handler (with the necessary subset of Host protocol) together into a single machine. Then, since certain new machines appeared attractive in cost/performance relative to the Honeywell 16 series, we spent considerable time thinking about what machine to use. We decided that no alternate choice was sufficiently attractive to justify rewriting the main IMP program and redesigning the Host and modem interface hardware. This left us with a decision between the Honeywell 516 and 316 computers. We chose the 316 on the basis of size and cost, feeling that the somewhat higher bandwidth of the 516 was not essential to the job and did not justify its higher price and the second cabinet which would have been required.

TERMINAL IMP HARDWARE

Figure 4 shows how the TIP fits the user into the network. Up to 63* terminals, either remote or local, of widely diverse characteristics may be connected to a given TIP and thereby "talk" into the network. It is also possible to connect a Host to a TIP in the usual way.

The TIP is shown in Figure 5. It is built around a Honeywell H-316 computer with 20K (20,480 words) of core. It embodies a standard 16 port multiplexed memory channel with priority interrupts and includes a Teletype for debugging and program reloading. Other features of the standard IMP also present are a real-time clock, power-fail and auto-restart mechanisms, and a program-generated interrupt feature.[2] As in the standard IMP, interfaces are provided for connecting to high-speed (50 kilobit, 230.4 kilobit, etc.) modems as well as to Hosts. The single-cabinet version limits the configuration to a total of three modem and/or Host interfaces, but an expansion cabinet may be used to increase this limit. More basic limits are set by the machine's logical organization (specifically the number of available memory channels) and the program bandwidth capability as discussed below.

Aside from the additional 8K of core memory, the primary hardware feature which distinguishes the TIP from a standard IMP is a Multi-Line Controller (MLC) which allows for connection of terminals to the

* There are 64 hardware lines but line 0 is logically reserved by the program for special use.

IMP. Any of the MLC lines may go to local terminals or via modems to remote terminals. As shown in Figure 6 the MLC consists of two portions, one a piece of central logic which handles the assembly and disassembly of characters and transfers them to and from memory, and the other a set of individual Line Interface Units (all identical except for small number of option jumpers) which synchronize reporting to individual data bits between the central logic and the terminal devices and provide for control and status information to and from the modem or device. Line Interface Units may be physically incorporated one at a time as required.

The MLC connects to the high-speed multiplexed memory channel option of the H-316, and uses three of its channels as well as two priority interrupts and a small number of control instructions. The MLC is fabricated by BBN and is built from TTL integrated circuits. The MLC central controller, complete with power supply, is housed in an H-316 expansion chassis, and the entire MLC, as well as the computer itself, is mounted in a standard six-foot rack. Additional space is provided in the bottom of the rack for up to sixteen card-mounted modems of the Bell 103, 201, or 202 variety, together with their power supplies.

In order to accommodate a variety of devices, the controller handles a wide range of data rates and character sizes, in both synchronous and asynchronous modes. Data characters of length 5-, 6-, 7-, or 8-bits are allowed by the controller. Since no interpretation of characters is done by the hardware, any character set, such as Baudot, Transcode ASCII or EBCDIC may be used.

The following is a list of data rates accepted by the controller:

SYNCHRONOUS	*ASYNCHRONOUS* (Nominal Rates)		
Any rate up to and including 19.2 Kb/s	75	1200	
	110	1800	
	134.5	2400	
	150	4800	output only
	300	9600	output only
	600	19200	output only

All above in bits/second

The data format required of all devices is bit serial and each character indicates its own beginning by means of a start bit which precedes the data and includes one or more stop bits at the end of the character. This per-character framing is quite standard for asynchronous lines but synchronous lines, generally designed for higher bandwidths, frequently adopt some form of "binary synchronous communication" where the characters are packed tightly together into messages which are then framed by special characters. Framing is thereby amortized over the entire message, thus consuming a smaller fraction of the available bandwidth than the per-character framing which uses two or more bits for every character. The difficulties with this scheme, however, are that it is more complex, requiring more sophisticated hardware at each end, and that no real standards exist which are adhered to by all or even most types of synchronous devices. We therefore decided to adopt per-character framing with start and stop bits even on synchronous lines. At a cost of some twenty percent of the bandwidth for framing, a very simple and general scheme is thus arrived at. A number of high speed terminal manufacturers, faced with the same problems, have arrived at a similar conclusion.

Given these characteristics, then, the controller will connect to the great majority of normal terminal devices such as Teletypes, alphanumeric CRT units, and modems, and also (with suitable remote interface units) to many peripheral devices such as card readers, line printers, and graphics terminals. Either full or half duplex devices can be accommodated. The standard TIP program cannot deal with a magnetic tape unit through the MLC. However, as a special option, and with the use of additional core memory, standard Honeywell tape drives can be connected to the TIP as normal peripherals.

The individual terminal line levels are consistent with EIA RS-232C convention. Data rates and character length are individually set for each line *by the the program*. For incoming asynchronous lines, the program includes the capability for detecting character length and line data rate as discussed below.

Logically, the controller consists of 64 input ports and 64 output ports. Each input/output pair is brought out to a single connector which is normally connected to a single device. However, by using a special "Y"

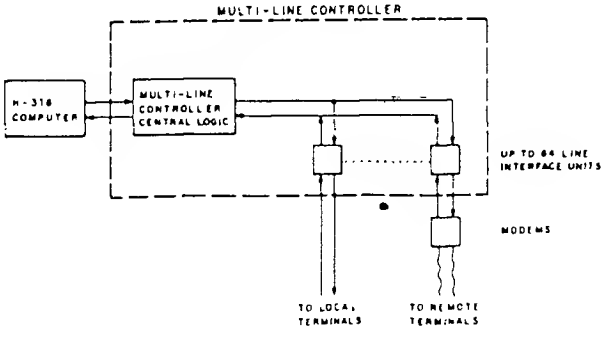


Figure 6—Block diagram of TIP hardware

eable, the ports may go to completely separate devices of entirely different properties. Thus, *input* port 16 may connect to a slow, asynchronous, 5-bit character keyboard while *output* port 16 connects to a high speed, synchronous display of some sort. In order to achieve this flexibility, the MLC stores information about each input and each output port and the program sets up this information for each half of each port in turn as it turns the ports "ON."

Several aspects of the MLC design are noteworthy. The central logic treats each of the 64 ports in succession, each port getting 800 ns of attention per cycle. The port then waits the remainder of the cycle (51.2 μs) for its next turn. For both input and output, two full characters are buffered in the central logic, the one currently being assembled or disassembled and one further character to allow for delays in memory accessing.

During input, characters from the various lines stream into a tumble table in memory on a first come, first served basis. Periodically a clock interrupt causes the program to switch tables and look for input.

Output characters are fed to all lines from a single output table. Ordering the characters in this table in such a way as to keep a set of lines of widely diverse speeds solidly occupied is a difficult task. To assist the program in this, a novel mechanism has been built into the MLC hardware whereby each line, as it uses up a character from the output table, enters a request consisting of its line number into a "request" table in memory. This table is periodically inspected by the program and the requests are used in building the next output table with the characters in proper line sequence.

The design of the terminal interface portion of the MLC is modular. Each Line Interface Unit (LIU) contains all the logic required for full duplex, bit serial communication and consists of a basic bi-directional data section and a control and status section. The data section contains transmit and receive portions each with clock and data lines. For asynchronous devices the clock line is ignored and timing is provided by the MLC itself. (For received asynchronous characters, timing is triggered by the leading edge of the start bit of each character.)

The control and status monitor functions are provided for modems as required by the RS-232C specification. Four outputs are available for control functions and six inputs are available to monitor status. The outputs are under program control and are available for non-standard functions if the data terminal is not a modem. For example, these lines could be used to operate a local line printer. RS-232C connectors are mounted directly on the LIU cards. To allow for variations in terminal and modem pin assignments, the signals are brought to connector pins via jumpers on the card.

The central MLC contains 256 ICs, many of which are MSI and some of which are LSI circuits, and it is thus about the same complexity as the basic H-316. In addition each LIU contains 31 ICs. A Terminal IMP including the MLC and with a typical interface configuration to high-speed circuits and Hosts is, order of magnitude, a $100,000 device.

THE TERMINAL USER'S VIEW

This section describes how a TIP user gains access to the network. The protocol described is of very recent origin and will undoubtedly change in response to usage which, as of this writing, is just commencing.

In general a user must have some foreknowledge of the resource which he expects to access via the network. The TIP program implements a set of commands[8] for connecting to and disconnecting from remote sites, but once a terminal is connected to a particular system, the TIP becomes transparent to the conversation unless specially signalled. This is equivalent to a time-sharing system where the executive program is essentially out of the picture during periods while the user is dealing directly with his own set of programs.

Because of the large number of different terminal types used in the network, the concept of the Network Virtual Terminal was developed. This is an imaginary but well-defined type of terminal. The TIP translates typed data to virtual terminal code before shipping it into the network, and conversely translates the remote system's response back into the local terminal's code. Thus, each Host system must deal only with this single terminal type.

When the user at a terminal needs to talk directly to his TIP instead of to the remote Host, he issues a command which is distinguished by the fact that it always starts with the symbol @. One or more words, perhaps followed by a single parameter, then identify the type of command.

Normally a user will go through four more or less distinct stages in typing into the net. First, he will be concerned with hardware, power, dialing in, etc. Then he will establish a dialogue with the TIP to get a comfortable set of parameters for his usage. Next, he will instruct the TIP to make a connection to a remote Host, and finally, he will mostly ignore the TIP as he talks to the remote Host.

One of the more interesting features of the TIP is that it permits great flexibility in the types of terminals which may be attached to any port. This

TABLE II—Tip Commands

CLOSE
close all outstanding connections

HOST #
focus attention on this host 0<#<256

LOGIN
start the initial connection procedure to get Telnet connections

SEND BREAK
send a Telnet break character

SEND SYNC
send a Telnet sync character and an INTERRUPT SENDER message

ECHO ALL
local TIP-generated echo—TIP echoes everything

ECHO NONE
remote Host-generated echo—TIP will echo commands

ECHO HALFDUPLEX
terminal-generated echo—TIP echoes nothing

FLUSH
delete all characters in input buffer

TRANSMIT EVERY #
send off input buffer at least every #th character 0<#<256

TRANSMIT ON EVERY CHARACTER
like TRANSMIT EVERY 1

TRANSMIT ON LINEFEED
send input buffer every time a linefeed encountered

TRANSMIT ON MESSAGE-END
send input buffer every time an EOM encountered

TRANSMIT ON NO CHARACTER
do not transmit on linefeed or EOM

TRANSMIT NOW
send off input buffer now

DEVICE RATE #
is a 13 bit code specifying hardware rate and character size settings

DEVICE CODE ASCII
DEVICE CODE 2741
DEVICE CODE EXECUPORT
DEVICE CODE ODEC
} establish code conversion

SEND TO HOST #
RECEIVE FROM HOST #
SEND TO SOCKET #
RECEIVE FROM SOCKET #
} establish parameters for manual initiation of connections

PROTOCOL TO TRANSMIT
PROTOCOL TO RECEIVE
PROTOCOL TO CLOSE TRANSMIT
PROTOCOL TO COSE RECEIVE
} initiate connection protocol manually

PROTOCOL BOTH
PROTOCOL TO CLOSE BOTH
} abbreviations for simultaneous transmit and receive protocols

GIVE BACK
release control of captured device

DIVERT OUTPUT
capture device # and divert this terminal's output to it

ABORT LOGIN
abort the outstanding initial connection procedure

flexibility presents a problem to the program which must determine what kind of terminal (speed, character size, etc.) is attached to a given port before a sensible exchange of information is possible. To solve this problem, each terminal is assigned a special identifying character which the user types repeatedly when starting to use a terminal. When information starts to appear from a previously idle port, the TIP enters a "hunt" mode in which it interprets the arriving characters trying various combinations of terminal characteristics. All except the proper combination will cause the character to be garbled, producing something other than one of the legal terminal identifying characters. When the TIP thus identifies the correct set (which generally requires the user to repeat the identifier less than half a dozen times), it types out 'HELLO in the terminal's own language.

In the next stage, the user initializes certain conversation parameters relating to message size and when to echo. He then establishes connection to a remote site using two commands which identify the desired remote site and the fact that the user wishes to be connected to the logger at that site. These commands are:

@ HOST 15

@ LOGIN

The LOGIN command actually sets in motion an elaborate exchange of messages between the TIP and the remote Host which normally result in a connection being made to that remote system. This command will be answered by an appropriate comment to the user indicating either that a connection has been made, that the remote site is not up, that it is up but actively refusing to converse, or that it is up but not responding enough even to refuse the connection.

Once a connection is established, the user types directly to the logger. The TIP does not execute the actual login since this procedure varies from site to site.

Throughout the user-to-Host dialogue, commands remain available at any time. Prior to closing the connection, the user must log out as required by the system he is using. He then gives the command

@ CLOSE

which causes the TIP to close the connection, informing the user when the process is finished.

In addition to the above more or less standard procedures, there are a number of less usual commands which set such things as device rate, character size, code types, etc. Such commands are used by a terminal on one port in setting parameters for a non-interactive terminal, such as a printer or card reader, on some

other port. Other special commands permit conversations directly between terminals on the same or different TIPs, allow for binary mode, etc. Table II is a list of the commands with a brief explanation. For details refer to Reference 8. Figure 7 gives an example of typical dialogue.

THE SOFTWARE

Because the terminals connected to a TIP communicate with Hosts at remote sites, the TIP, in addition to performing the IMP function, also acts as intermediary between the terminal and the distant Host. This means that network standards for format and protocol must be implemented in the TIP. One can thus think of the TIP software as containing both a very simpleminded mini-Host and a regular IMP program.

Figure 8 gives a simplified diagrammatic view of the program. The lower block marked "IMP" represents the usual IMP program. The two lines into and out of that block are logically equivalent to input and output from a Host. The code conversion blocks are in fact surprisingly complex and include all of the material for dealing with diverse (and perverse) types of terminals.

As the user types on the keyboard, characters go, via input code conversion, to the input block. Information for remote sites is formed into regular network messages and passes through the OR switch to the IMP program for transmittal. Command characters are fed off to the side to the command block where commands are decoded. The commands are then "per-

1	Character typed just after terminal turned on. Used to identify the terminal type for the Terminal IMP program.	
HELLO	TIP indicates that terminal is ready for use.	
@ HOST 1	User tells Terminal IMP which Host to connect to.	
	(Terminal IMP echoes extra carriage return line feed to indicate it has accepted a command.)	
@ LOGIN	User tells Terminal IMP to form a connection.	
	(TIP echoes extra carriage return line feed.)	
? R OPEN	Terminal IMP has formed a connection.	
LOG ON"	This is UCLA I7's greeting message.	Dialogue between terminal user and UCLA I7.
IIII	User identification.	
!	.	
$" ,ABACUS,C	.	
.	.	
.	.	
.	.	
*	.	
STOP	.	
NORMAL EXIT	.	
!	.	
*	User exits from UCLA I7.	
LOG ON"	I7 closing message indicates user has logged off.	
@ CLOSE	User detaches the Terminal IMP from computer being used (i.e. closes connection).	
	(TIP echoes extra carriage return line feed.)	
? R CLOSED	Terminal IMP has closed the connection.	

Figure 7—Typical terminal user dialogue

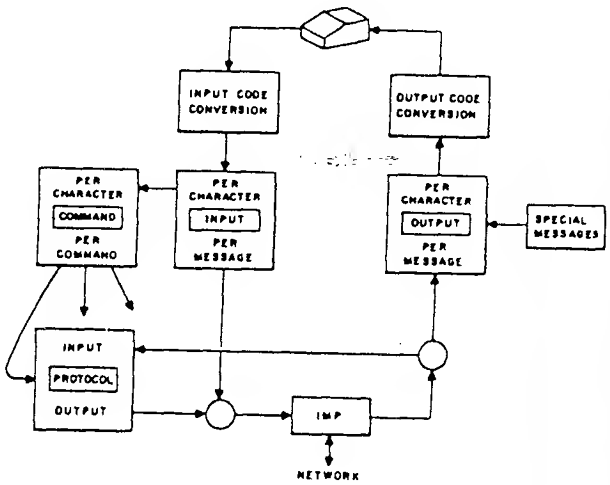


Figure 8—Block diagram of TIP program

formed" in that they either set some appropriate parameter or a flag which calls for later action. An example of this is the LOGIN command. Such a command in fact triggers a complex network protocol procedure, the various steps of which are performed by the PROTOCOL block working in conjunction with the remote Host through the IMPs. As part of this process an appropriate special message will be sent to the terminal via the Special Messages block indicating the status (success, failure, etc.) of the procedure.

Once connection to a remote Host is established, regular messages flow directly through the Input block and on through the IMP program. Returning responses come in through the IMP, into the OUTPUT program where they are fed through the OUTPUT Code Conversion block to the terminal itself.

Storage for the TIP program, including the standard IMP program, is just over 20K. This is roughly as follows:

Standard IMP Program with buffers & tables	12,000
Special TIP code	2,650
Tables of Parameters	1,800
Buffer Storage for messages to and from terminals	1,880
Miscellaneous—I/O buffers, constants, etc.	1,880

PERFORMANCE

The program can handle approximately 100 kilobits of one-way terminal traffic provided the message size

is sufficiently large that per-message processing is amortized over many characters. Overhead per message is such that if individual characters are sent one at a time there is a loss of somewhere between a factor of ten and twenty in bandwidth. A different way to look at program performance is to observe that the per-character processing time is about 75 μs.

These figures ignore the fact that the machine must devote some of its bandwidth to acting as an IMP, both for terminal traffic and for regular network traffic. About 5% of the machine is lost to acting as an IMP, even in the absence of traffic. If there is network traffic, more of the machine bandwidth is used up. Five hundred kilobits of two-way phone line and Host traffic saturates the machine without any terminal traffic*.

In addition to bandwidth which goes into the IMP part of the job, another 10 percent of the (total) machine is taken up simply in fielding clock interrupts from the Multi-Line Controller. This again is bandwidth used in idling even with no actual terminal traffic.

The following formula summarizes, approximately, the bandwidth capabilities:

$$P + H + 11T \leq 850$$

where:

- P = total phone line traffic (in kilobits/sec) wherein, for example, a full duplex 50 Kb phone line counts as 100;
- H = total Host traffic (in kilobits/sec) wherein the usual full duplex Host interface, with its usual 10 μs/bit setting, counts as 200; and
- T = total terminal traffic (in kilobits/sec) wherein an ASCII terminal such as a (110-baud) full-duplex Teletype (ASR-33) counts as twice its baud rate (i.e., 0.220 Kb).

This means that it takes eleven times as much program time to service every terminal character as it does to service a character's worth of phone line or Host message.

A further factor that influences terminal traffic handling capability has to do with the terminals themselves. Certain types of terminals require more attention from the program than others, independent of their speed but based rather on their complexity. In particular, for example, while an IBM 2741 nominally runs at 134.5 bits per second, the complexity is such that it uses nearly three times the program bandwidth that would be used in servicing a half-duplex ASCII terminal of equivalent speed. Allowances for such variations must be made in computing the machine's ability to service a particular configuration. It must be borne in mind that all of these performance figures are approximations and that the actual rules are extremely complex, and will change as the program matures.

* The number 500 kilobits is for full size (8000 bit) messages. Shorter messages use up more capability per bit and thus reduce the overall bandwidth capability.

DISCUSSION

As the ARPA Network grows, a number of areas of development seem likely to require attention. Certain of these are already pressing problems whereas others will begin to appear primarily as the network continues to grow and mature.

Perhaps the most difficult aspect of a network such as this is the problem of reliability. A great deal of thought and effort has gone into trying to make the system reliable and network topology has been designed with the need for redundancy in mind. Nonetheless the problem of keeping a large number of computer systems going at widely separated sites is non-trivial. An IMP's mean-time-between-failure (MTBF) has been on the order of one month; considerably lower than expected. The problems arise from a variety of sources and the system is sufficiently complex that frequently the cause has been masked by the time the failure has been noted. Often the same failure will recur several times until the problem is identified and eliminated and this fact tends to decrease the apparent MTBF. In general a preponderance of the troubles stem from hardware failures, and we are currently modifying several noisy and/or marginal portions of the I/O and interface hardware in hopes of obtaining significant improvement.

Our strategy has generally been to spend time identifying the source of trouble, keeping the machine off the air if necessary, so that eventually the MTBF will increase. This has often meant, however, that a "down" was much longer than it would have been, had the foremost goal been to get the machine running again immediately. With this strategy the average down time has been about 9 hours, giving rise to an average per-machine down rate of about 2 percent. We hope to improve this situation in the near future by providing improved facilities for obtaining "program dumps" when a failure occurs. This will make it possible to bring machines back on the air almost immediately in many cases, without jeopardizing valuable debugging data.

A word about our experience with the phone lines appears appropriate here as well. We have apparently been an unusual, if not unique, customer for the common carriers in our degree of attention to line failures. Our ability to detect and report even brief outages has led through measured skepticism to eventual acceptance by the carriers. In general, tests have indicated that the phone line error rates are about as predicted, i.e., approximately one in 10^6. These occasional errors or error bursts do not appear to affect network performance adversely. The IMP-to-IMP error checking procedure has not, to our knowledge, admitted a single erroneous message. We have, however, had some difficulty with lines which were simply out of commission from time to time for extended periods (hours or even days). The reliability of the phone lines is roughly equivalent to the reliability of the IMPs, based on the number and duration of outages. Down times have decreased as the carriers have come to accept our complaints as generally legitimate. *Overall the performance of the telephone equipment does not appear to constitute a problem in network growth.*

From a strictly technical viewpoint we view the incorporation of higher bandwidth facilities as a natural and key part of network growth. While the present facilities are not saturated by the present loads, we view this situation as a temporary one and something of a period of grace. As the network continues to grow over the next few years traffic can be expected to increase in a very non-linear fashion. Host protocol procedures (which have presented a sizable stumbling block to usage) are settling down so that software commitments can be made, and the advent of the Terminal IMP will bring an influx of new users who do not have the alternative of local resources. As a result we expect that traffic will begin to saturate some parts of the network. Terminal IMPs may well be called upon to service a larger number of terminals of higher bandwidth than can be handled by the present version.

In anticipation of these requirements we are presently considering the design of a significantly faster and more modular version of the IMP. Its higher speed will permit it to take advantage of high speed (i.e., megabit and above) communication lines which the common carriers are currently developing. More generally it will be able to service high bandwidth requirements whenever and however they occur within the net. We are currently studying a modular design which will permit connection of a greater number of interfaces than the present IMP can handle. While this work is only in the preliminary design stage, we feel that the interest and enthusiasm which have greeted the initial network suggest that it is not too early to consider ways to cope with growing traffic requirements.

Another area of network development which has already received a great deal of attention and will require much more in the future is that of technical information interchange between the sites. To the user, the resources which are becoming available are staggering if not overwhelming. It is very difficult for an individual to discover what, if any, of the mass of programs that will be available through the network may be of interest or of use to him. To aid in this process a number of mechanisms are being implemented and we are cooperating closely with these efforts. In particular, at the Stanford Research Institute a Network Information Center (NIC) acts essentially as a library for documentation concerning the network. Here are maintained a number of pointer documents, specifically, (1) a Directory of Participants which lists critical personnel, phone numbers, etc., for each participating site, (2) a catalogue of the NIC Collection which lists the available documents indexed in a variety of ways, (3) a Resource Notebook which is a compendium that attempts to familiarize the reader with available program resources at each of the participating sites.

Finally, there is the broad and exciting issue of how to cope with success. As the ARPA Network grows, and as diverse resources and users join the net, it is clear that a technology transfer must occur; the network probably must shift from a government-supported research and development activity to an ongoing national *service* of some kind. However, the computer-communications environment in the U.S. is rather complex, and is populated with many legal, economic, and political constraints. Within this environment, it is not easy to perform the technology transfer, and many groups, including ARPA and BBN, have been considering possible alternative plans. We are optimistic that a way will be found to provide a suitable legal and political base for expansion of the present ARPA Network into a widely useful public communication system.

ACKNOWLEDGMENTS

In addition to the authors, a great many people have contributed to the work reported here. Dr. L. G. Roberts of the Advanced Research Projects Agency has continued to lend crucial support and encouragement to the project. Messrs. Blue, Dolan, and Crocker of the ARPA office have been understanding and helpful. Suggestions and helpful criticism have come from many present and prospective participants in the network community.

At BBN, W. B. Barker is responsible for much of the eleverness which appears in the MLC hardware; R. E. Kahn has contributed in many areas and has been deeply involved in studying traffic flow control; A. McKenzie has compiled the Network Resource Notebook and been concerned with Host relations and many Host protocol issues; M. Thrope has managed the Network Control Center and has kept the network on the air; J. Levin helped in implementation of the TIP software; B. P. Cosell has patiently labored with the IMP software, with help from J. McQuillan and M. Kraley; and R. Alter has offered much helpful advice and criticism.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

2 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

5 C S CARR S D CROCKER V G CERF
Host-host communication protocol in the ARPA network
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

6 S CROCKER et al
Function-oriented protocols for the ARPA computer network
AFIPS Conference Proceedings Spring Joint Computer Conference 1972

7 R E KAHN W R CROWTHER
Flow control in a resource sharing computer network
Proc Second Symposium on Problems in the Optimization of Data Communications Systems October 1971

8 *User's guide to the terminal IMP*
Bolt Beranek and Newman Inc Report No 2183

9 *The BBN terminal interface message processor*
BBN Report 2184

10 L G ROBERTS B D WESSLER
The ARPA network
Computer Communication Networks Chapter 6 In preparation

11 L G ROBERTS
A forward look
Journal of Armed Forces Communications and Electronics Assoc Vol XXV No 12 August 1971 pp 77-81

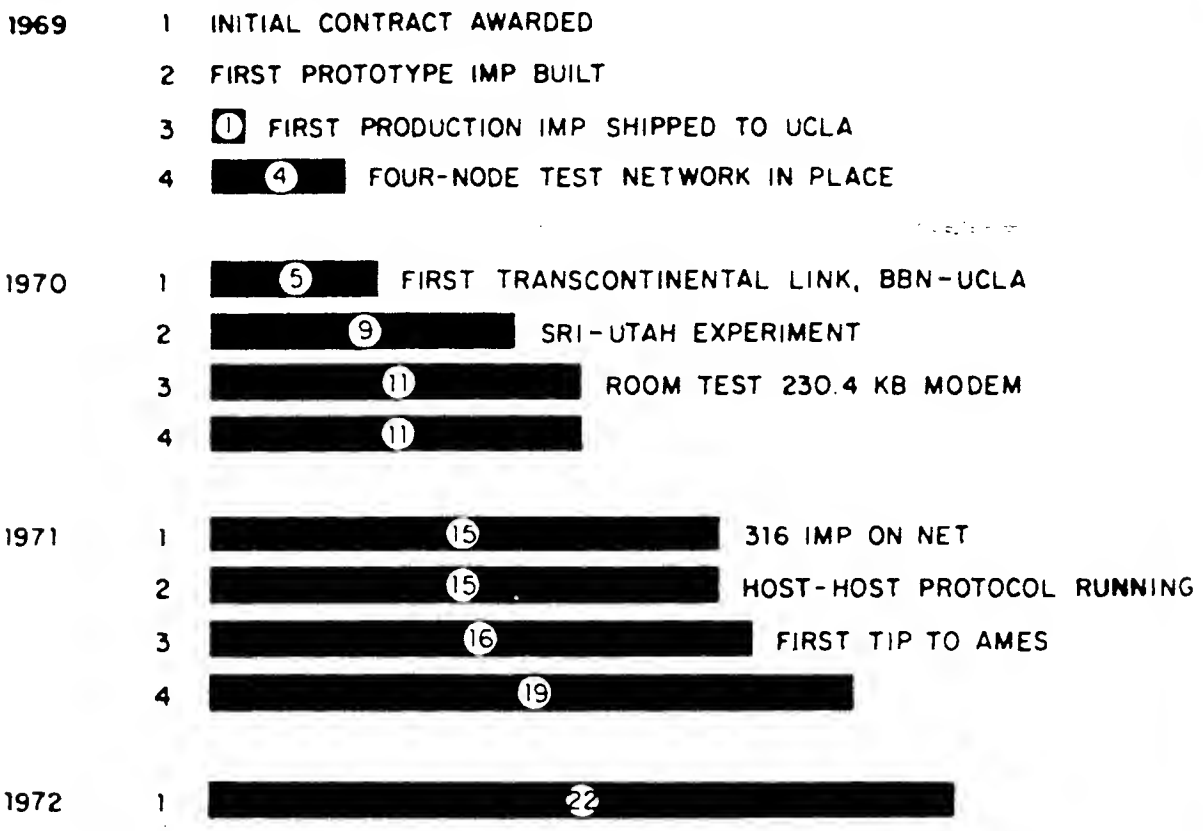
12 *The MIT Lincoln Lab terminal support processor*
Graphics Semi-Annual Tech Summary Reports ESD-TR-70-151 p 355 May November 1970

13 *Progress report #1*
University of Michigan MERIT Computer Network Report #0571PR4 May 1971

14 A B COCANOWER W FISCHER W S GERSTENBERGER B S READ
The communications computer operating system—The initial design
University of Michigan MERIT Computer Network Manual #1070-TN-3 October 1970

15 R E KAHN
Terminal access to the ARPA computer network
Courant Computer Symposium 3 Computer Networks Courant Institute New York November 1970—Proceedings to be published by Prentice Hall Englewood Cliffs New Jersey In preparation

QUARTER — NUMBER OF NODES ON NET / MILESTONES



HOST COMPUTER TYPES • NUMBER

PDP-1 · 2	TSP-1 · 1	360/44 · 1
PDP-10 · 13	TX-2 · 1	360/65 · 1
PDP-11 · 3	B6700 · 1	360/67 · 3
H316 · 1	UNIVAC 418 III · 2	360/75 · 1
H645 · 1	SIGMA 7 · 1	360/91 · 1

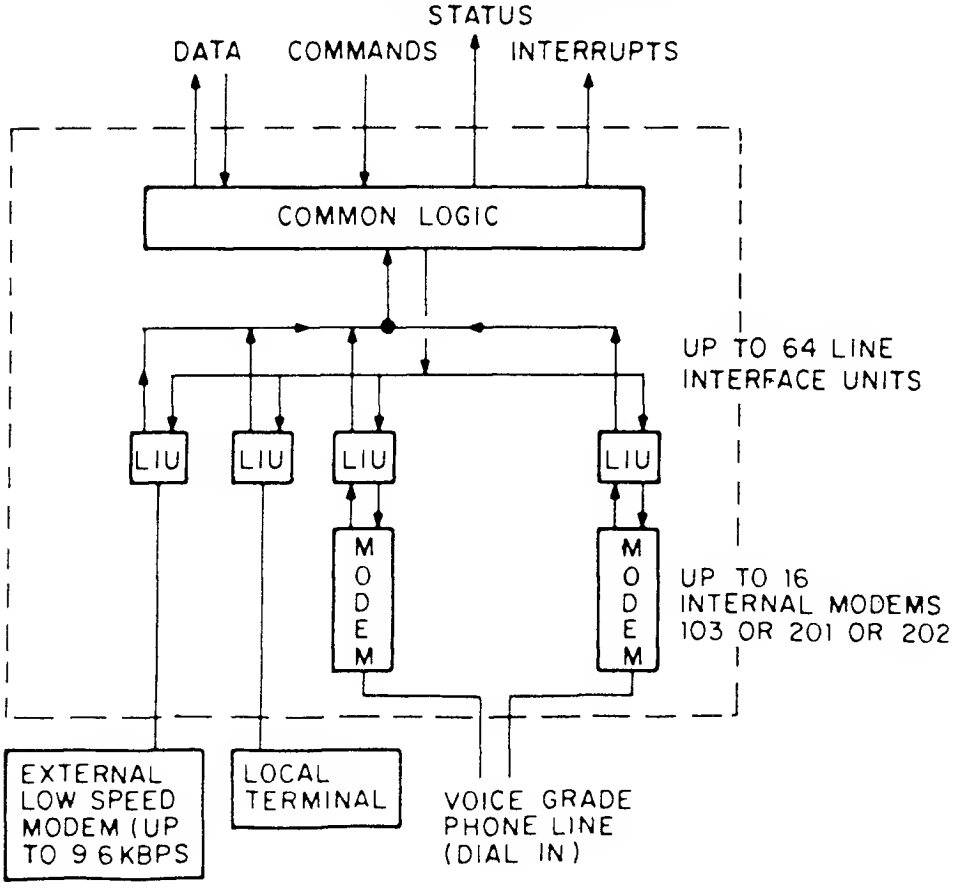
MULTI-LINE CONTROLLER CHARACTERISTICS

- 64 LINES
- CLOCKED / UNCLOCKED
- START / STOP FORMAT
- CHARACTER SIZE (5-8 BITS)
- SPEEDS

  CLOCKED → ANY UP TO 19.2 KB

  UNCLOCKED → ALL STANDARD SPEEDS
- FULL OR HALF DUPLEX
- TERMINALS OF MANY TYPES
- PER LINE PROGRAM - SETTABLE CHACTERISTICS (INCLUDING CROSSPATCHING)

MULTI-LINE CONTROLLER



Computer communication network design—Experience with theory and practice*

by HOWARD FRANK
Network Analysis Corporation
Glen Cove, New York

ROBERT E. KAHN
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

and

LEONARD KLEINROCK
University of California
Los Angeles, California

INTRODUCTION

The ARPA Network (ARPANET) project brought together many individuals with diverse backgrounds, philosophies, and technical approaches from the fields of computer science, communication theory, operations research and others. The project was aimed at providing an efficient and reliable computer communications system (using message switching techniques) in which computer resources such as programs, data, storage, special purpose hardware etc., could be shared among computers and among many users.[38] The variety of design methods, ranging from theoretical modeling to hardware development, were primarily employed independently, although cooperative efforts among designers occurred on occasion. As of November, 1971, the network has been an operational facility for many months, with about 20 participating sites, a network information center accessible via the net, and well over a hundred researchers, system programmers, computer center directors and other technical and administrative personnel involved in its operation.

In this paper, we review and evaluate the methods used in the ARPANET design from the vantage of over two years' experience in the development of the network. In writing this paper, the authors have each made equal contributions during a series of intensive discussions and debates. Rather than present merely a summary of the procedures that were used in the network design, we have attempted to evaluate each other's methods to determine their advantages and drawbacks. Our approaches and philosophies have often differed radically and, as a result, this has not been an easy or undisturbing process. On the other hand, we have found our collaboration to be extremely rewarding and, notably, we have arrived at many similar conclusions about the network's behavior that seem to be generally applicable to message switched networks.

The essence of a network is its design philosophy, its performance characteristics, and its cost of implementation and operation. Unfortunately, there is no generally accepted definition of an "optimal" network or even of a "good" network. For example, a network designed to transmit large amounts of data only during late evening hours might call for structural and performance characteristics far different from one servicing large numbers of users who are rapidly exchanging short messages during business hours. We expect this topic, and others such as the merits of message switching vs. circuit switching or distributed vs. centralized control to be a subject of discussion for many years.[1,14,24,32,34,37]

A cost analysis performed in 1967-1968 for the ARPA Network indicated that the use of message switching would lead to more economical communications and better overall availability and utilization of resources than other methods.[36,38] In addition to its impact on the availability of computer resources, this decision has generated widespread interest in store-and-forward communications. In many instances, the use of store-and-forward communication techniques can result in

* This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC 15-70-C-0120 at the Network Analysis Corporation, Contract Nos. DAHC 15-69-C-0179 and DAHC-71-C-0088 at Bolt Beranek and Newman Inc., and Contract No. DAHC 15-69-C-0285 at the University of California at Los Angeles.

greater flexibility, higher reliability, significant technical advantage, and substantial economic savings over the use of conventional common carrier offerings. An obvious trend toward increased computer and communication interaction has begun. In addition to the ARPANET, research in several laboratories is under way, small experimental networks are being built, and a few examples of other government and commercial networks are already apparent.[6,7,31,40,41,47,48,52]

In the ARPANET, each time-sharing or batch processing computer, called a Host, is connected to a small computer called an Interface Message Processor (IMP). The IMPs, which are interconnected by leased 50 kilobit/second circuits, handle all network communication for their Hosts. To send a message to another Host, a Host precedes the text of its message with an address and simply delivers it to its IMP. The IMPs then determine the route, provide error control, and notify the sender of its receipt. The collection of Hosts, IMPs, and circuits forms the message switched resource sharing network. A good description of the ARPA[illegible], and some early results on protocol development and modeling are given in References 3, 12, 15, 23 and 38. Some experimental utilization of the ARPANET is described in Reference 42. A more recent evaluation of such networks and a forward look is given in References 35 and 39.

The development of the Network involved four principal activities:

(1) The design of the IMPs to act as nodal store-and-forward switches,
(2) The topological design to specify the capacity and location of each communication circuit within the network,
(3) The design of higher level protocols for the use of the network by time-sharing, batch processing and other data processing systems, and
(4) System modeling and measurement of network performance.

Each of the first three activities were essentially performed independently of each other, whereas the modeling effort partly affected the IMP design effort, and closely interacted with the topological design project.

The IMPs were designed by Bolt Beranek and Newman Inc. (BBN) and were built to operate independent of the exact network connectivity; the topological structure was specified by Network Analysis Corporation (NAC) using models of network performance developed by NAC and by the University of California at Los Angeles (UCLA). The major efforts in the area of system modeling were performed at UCLA using theoretical and simulation techniques. Network performance measurements have been conducted during the development of the network by BBN and by the Network Measurement Center at UCLA. To facilitate effective use of the net, higher level (user) protocols are under development by a group of representatives of universities and research centers. This group, known as the Network Working Group, has already specified a Host to Host protocol and a Telnet protocol, and is in the process of completing other function oriented protocols.[4,29] We make no attempt to elaborate on the Host to Host protocol design problems in this paper.

THE NETWORK DESIGN PROBLEM

A variety of performance requirements and system constraints were considered in the design of the net. Unfortunately, many of the key design objectives had to be specified long before the actual user requirements could be known. Once the decision to employ message switching was made, and fifty kilobit/second circuits were chosen, the critical design variables were the network operating procedure and the network topology; the desired values of throughput, delay, reliability and cost were system performance and constraint variables. Other constraints affected the structure of the network, but not its overall properties, such as those arising from decisions about the length of time a message could remain within the network, the location of IMPs relative to location of Hosts, and the number of Hosts to be handled by a single IMP.

In this section, we identify the central issues related to IMP design, topological design, and network modeling. In the remainder of the paper, we describe the major design techniques which have evolved.

IMP properties

The key issue in the design of the IMPs was the definition of a relationship between the IMP subnet and the Hosts to partition responsibilities so that reliable and efficient operation would be achieved. The decision was made to build an autonomous subnet, independent (as much as possible) of the operation of any Host. The subnet was designed to function as a "communications system"; issues concerning the use of the subnet by the Hosts (such as protocol development) were initially left to the Hosts. For reliability, the IMPs were designed to be robust against all line failures and the vast majority of IMP and Host failures. This decision required routing strategies that dynamically adapt to changes in the states of IMPs and circuits,

and an elaborate flow control strategy to protect the subnet against Host malfunction and congestion due to IMP buffer limitations. In addition, a statistics and status reporting mechanism was needed to monitor the behavior of the network.

The number of circuits that an IMP must handle is a design constraint directly affecting both the structure of the IMP and the topological design. The speed of the IMP and the required storage for program and buffers depend directly upon the total required processing capacity, which must be high enough to switch traffic from one line to another when all are fully occupied. Of great importance is the property that all IMPs operate with identical programs. This technique greatly simplifies the problem of network planning and maintenance and makes network modifications easy to perform.

The detailed physical structure of the IMP is not discussed in this paper.[2,15] However, the operating procedure, which guides packets through the net, is very much of interest here. The flow control, routing, and error control techniques are integral parts of the operating procedure and can be studied apart from the hardware by which they are implemented. Most hardware modifications require changes to many IMPs already installed in the field, while a change in the operating procedure can often be made more conveniently by a change to the single operating program common to all IMPs, which can then be propagated from a single location via the net.

Topological properties

The topological design resulted in the specification of the location and capacity of all circuits in the network. Projected Host—Host traffic estimates were known at the start to be either unreliable or wrong. Therefore, the network was designed under the assumption of equal traffic between all pairs of nodes. (Additional superimposed traffic was sometimes included for those nodes with expectation of higher traffic requirements.) The topological structure was determined with the aid of specially developed heuristic programs to achieve a low cost, reliable network with a high throughput and a general insensitivity to the exact traffic distribution. Currently, only 50 kilobit/second circuits are being used in the ARPANET. This speed line was chosen to allow rapid transmission of short messages for interactive processing (e.g., less than 0.2 seconds average packet delay) as well as to achieve high throughput (e.g., at least 50 kilobits/second) for transmission of long messages. For reliability, the network was constrained to have at least two independent paths between each pair of IMPs.

The topological design problem requires consideration of the following two questions:

(1) Starting with a given state of the network topology, what circuit modifications are required to add or delete a set of IMPs?
(2) Starting with a given state of network topology, when and where should circuits be added or deleted to account for long term changes in network traffic?

If the locations of all network nodes are known in advance, it is clearly most efficient to design the topological structure as a single global effort. However, in the ARPANET, as in most actual networks, the initial designation of node locations is modified on numerous occasions. On each such occasion, the topology can be completely reoptimized to determine a new set of circuit locations.

In practice, there is a long lead time between the ordering and the delivery of a circuit, and major topological modifications cannot be made without substantial difficulty. It is therefore prudent to add or delete nodes with as little disturbance as possible to the basic network structure consistent with overall economical operation. Figure 1 shows the evolution of the ARPANET from the basic four IMP design in 1969 to the presently planned 27 IMP version. Inspection of the 24 and 27 IMP network designs reveals a few substantial changes in topology that take advantage of the new nodes being added. Surprisingly enough, a complete "reoptimization" of the 27 IMP topology yields a network only slightly less expensive (about 1 percent) than the present network design.[28]

Network models

The development of an accurate mathematical model for the evaluation of time delay in computer networks is among the more difficult of the topics discussed in this paper. On the one hand, the model must properly reflect the relevant features of the network structure and operation, including practical constraints. On the other hand, the model must result in a mathematical formulation which is tractable and from which meaningful results can be extracted. However, the two requirements are often incompatible and we search for an acceptable compromise between these two extremes.

The major modeling effort thus far has been the study of the behavior of networks of queues.[21] This emphasis is logical since in message switched systems, messages experience queueing delays as they pass from node to node and thus a significant performance measure is the

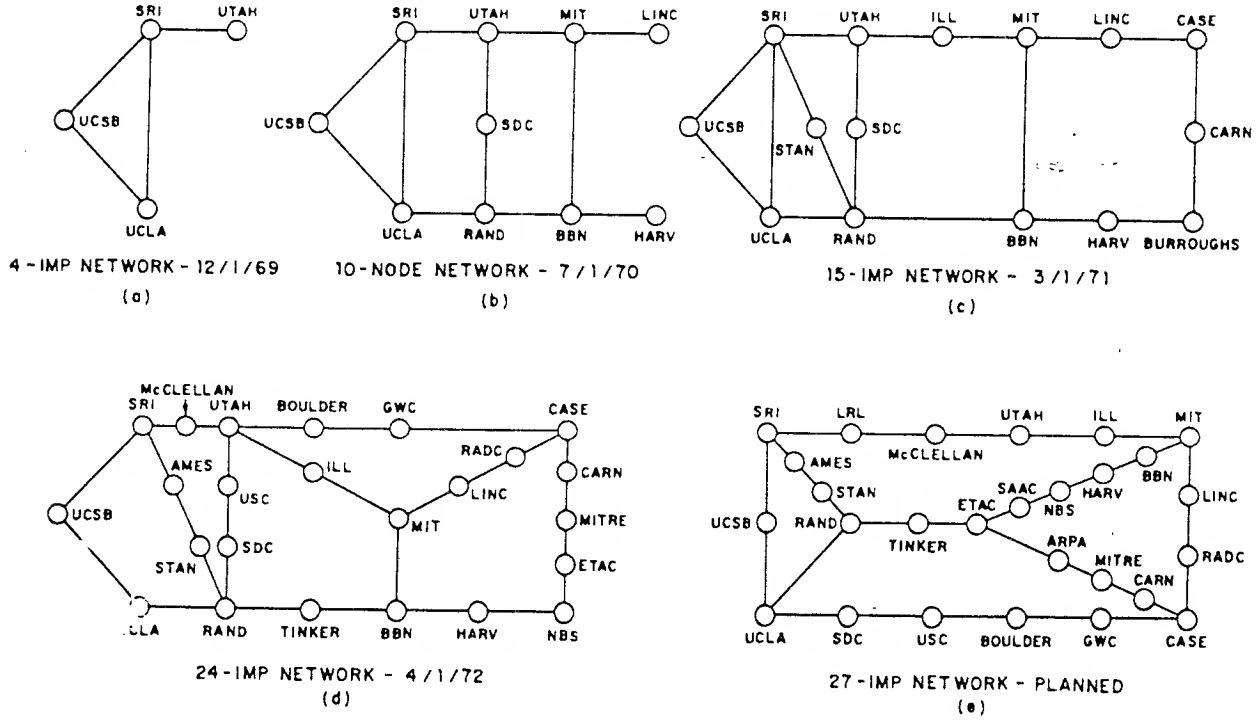


Figure 1—The evolution of the ARPANET

speed at which messages can be delivered. The queueing models were developed at a time when there were no operational networks available for experimentation and model validation, and simulation was the only tool capable of testing their validity. The models, which at all times were recognized to be idealized statements about the real network, were nonetheless crucial to the ARPANET topological design effort since they afforded the only known way to quantitatively predict the properties of different routing schemes and topological structures. The models have been subsequently demonstrated to be very accurate predictors of network throughput and indispensable in providing analytical insight into the network's behavior.

The key to the successful development of tractable models has been to factor the problem into a set of simpler queueing problems. There are also heuristic design procedures that one can use in this case. These procedures seem to work quite well and are described later in the paper. However, if one specializes the problem and removes some of the real constraints, theory and analysis become useful to provide understanding, intuition and design guidelines for the original constrained problem. This approach uncovers global properties of network behavior, which provide keys to good heuristic design procedures and ideal performance bounds.

DESIGN TECHNIQUES

In this section we describe the approaches taken to the design problems introduced in the previous section. We first summarize the important properties of the ARPANET design:

(1) A communications cost of less than 30 cents per thousand packets (approximately a megabit).
(2) Average packet delays under 0.2 seconds through the net.
(3) Capacity for expansion to 64 IMPs without major hardware or software redesign.
(4) Average total throughput capability of 10-15 kilobits/second for all Hosts at an IMP.
(5) Peak throughput capability of 85 kilobits/second per pair of IMPs in an otherwise unloaded network.
(6) Transparent communications with maximum message size of approximately 8000 bits and error rates of one bit in 10^{12} or less.

(7) Approximately 98 percent availability of any IMP and close to 100 percent availability of all operating IMPs from any operable IMP.

The relationships between the various design efforts are illustrated by these properties. The topological design provides for both a desired average throughput and for two or more paths to be fully used for communication between any pair of Hosts. The operating procedure should allow any pair of Hosts to achieve those objectives. The availability of IMPs to communicate reflects both the fact that IMPs are down about 2 percent of the time and that the topology is selected so that circuit failures contribute little additional to the total system downtime.

IMP design

The IMP design consists of two closely coupled but nonetheless separable pieces—the physical hardware specification (based on timing and reliability considerations and the operating procedure) and the design and implementation of the operating procedure using the specified IMP hardware. The IMP originally developed for the ARPANET contains a 16-bit one microsecond computer that can handle a total of about ¾ megabits/second of "useful" information on a total of approximately one megabit/second of circuit capacity (e.g., twenty 50 kilobit/second circuits). Hardware is likely to change as a function of the required IMP capacity but an operating procedure that operates well at one IMP capacity is likely to be transferable to machines that provide different capacity. However, as a network grows in size and utilization, a more comprehensive operating procedure that takes account of known structural properties, such as a hierarchical topology, is appropriate.

Four primary areas of IMP design, namely message handling and buffering, error control, flow control, and routing are discussed in this section. The IMP provides buffering to handle messages for its Host and packets for other IMPs. Error control is required to provide reliable communication of Host messages in the presence of noisy communication circuits. The design of the operating procedure should allow high throughput in the net under heavy traffic loads. Two potential obstacles to achieving this objective are: (1) The net can become congested and cause the throughput to *decrease* with increasing load, and (2) The routing procedure may be unable to always adapt sufficiently fast to the rapid movement of packets to insure efficient routing. A flow control and routing procedure is needed that can efficiently meet this requirement.

Message handling and buffering

In the ARPANET, the maximum message size was constrained to be approximately 8000 bits. A pair of Hosts will typically communicate over the net via a sequence of transmitted messages. To obtain delays of a few tenths of a second for such messages and to lower the required IMP buffer storage, the IMP program partitions each message into one or more packets each containing at most approximately 1000 bits. Each packet of a message is transmitted independently to the destination where the message is reassembled by the IMP before shipment to that destination Host. Alternately, the Hosts could assume the responsibility for reassembling messages. For an asynchronous IMP-Host channel, this marginally simplifies the IMP's task. However, if *every* IMP-Host channel were synchronous, and the Host provided the reassembly, the IMP task can be further simplified. In this latter case, "IMP-like" software would have to be provided in each Host.

The method of handling buffers should be simple to allow for fast processing and a small amount of program. The number of buffers should be sufficient to store enough packets for the circuits to be used to capacity; the size of the buffers may be intuitively selected with the aid of simple analytical techniques. For example, fixed buffer sizes are useful in the IMP for simplicity of design and speed of operation, but inefficient utilization can arise because of variable length packets. If each buffer contains A words of overhead and provides space for M words of text, and if message sizes are uniformly distributed between 1 and L, it can be shown[46] that the choice of M that minimizes the expected storage is approximately $\sqrt{AL}$. In practice, M is chosen to be somewhat smaller on the assumption that most traffic will be short and that the amount of overhead can be as much as, say, 25 percent of buffer storage.

Error control

The IMPs must assume the responsibility for providing error control. There are four possibilities to consider:

(1) Messages are delivered to their destination out of order.

(2) Duplicate messages are delivered to the destination.

(3) Messages are delivered with errors.

(4) Messages are not delivered.

The task of proper sequencing of messages for delivery to the destination Host actually falls in the province of both error control and flow control. If at most one message at a time is allowed in the net between a pair of Hosts, proper sequencing occurs naturally. A duplicate packet will arrive at the destination IMP after an acknowledgment has been missed, thus causing a successfully received packet to be retransmitted. The IMPs can handle the first two conditions by assigning a sequence number to each packet as it enters the network and processing the sequence number at the destination IMP. A Host that performs reassembly can also assign and process sequence numbers and check for duplicate packets. For many applications, the order of delivery to the destination is immaterial. For priority messages, however, it is typically the case that fast delivery requires a perturbation to the sequence.

Errors are primarily caused by noise on the communication circuits and are handled most simply by error detection and retransmission between each pair of IMPs along the transmission path. This technique requires extra storage in the IMP if either circuit speeds or circuit lengths substantially increase. Failures in detecting errors can be made to occur on the order of years to centuries apart with little extra overhead (20-30 parity bits per packet with the 50 kilobit/second circuits in the ARPANET). Standard cyclic error detection codes have been usefully applied here.

A reliable system design insures that each transmitted message is accurately delivered to its intended destination. The occasional time when an IMP fails and destroys a useful in-transit message is likely to occur far less often than a similar failure in the Hosts and has proven to be unimportant in practice, as are errors due to IMP memory failures. A simple end to end retransmission strategy will protect against these situations, if the practical need should arise. However, the IMPs are designed so that they can be removed from the network without destroying their internally stored packets.

Flow control

A network in which packets may freely enter and leave can become congested or logically deadlocked and cause the movement of traffic to halt.[5,17] Flow control techniques are required to prevent these conditions from occurring. The provision of extra buffer storage will mitigate against congestion and deadlocks, but cannot in general prevent them.

The sustained failure of a destination Host to accept packets from its IMP at the rate of arrival will cause the net to fill up and become congested. Two kinds of logical deadlocks, known as reassembly lockup and store-and-forward lockup may also occur. In reassembly lockup, the remaining packets of partially reassembled messages are blocked from reaching the destination IMP (thus preventing the message from being completed and the reassembly space freed) by other packets in the net that are waiting for reassembly space at that destination to become free. In a store-and-forward lockup, the destination has room to accept arriving packets, but the packets interfere with each other by tying up buffers in transit in such a way that none of the packets are able to reach the destination.[17] These phenomena have only been made to occur during very carefully arranged testing of the ARPANET and by simulation.[49]

In the original ARPANET design, the use of software links and RFNMS protected against congestion by a single link or a small set of links. However, the combined traffic on a large number of links could still produce congestion. Although this strategy did not protect against lockup, the method has provided ample protection for the levels of traffic encountered by the net to date.

A particularly simple flow control algorithm that augments the original IMP design to prevent congestion and lockup is also described in Reference 17. This scheme includes a mechanism whereby packets may be discarded from the net at the destination IMP when congestion is about to occur, with a copy of each discarded packet to be retransmitted a short time later by the originating Host's IMP. Rather than experience excessive delays within the net as traffic levels are increased, the traffic is queued outside the net so that the transit time delays internal to the net continue to remain small. This strategy prevents the insertion of more traffic into the net than it can handle.

It is important to note the dual requirement for small delays for interactive traffic and high bandwidth for the fast transfer of files. To allow high bandwidth between a pair of Hosts, the net must be able to accept a steady flow of packets from one Host and at the same time be able to rapidly quench the flow at the entrance to the source IMP in the event of imminent congestion at the destination. This usually requires that a separate provision be made in the algorithm to protect short interactive messages from experiencing unnecessarily high delays.

Routing

Network routing strategies for distributed networks require routing decisions to be made with only information available to an IMP and the IMP must

execute those decisions to effect the routing.[14,15] A simple example of such a strategy is to have each IMP handling a packet independently route it along its current estimate of the shortest path to the destination.

For many applications, it suffices to deal with an idealized routing strategy which may not simulate the IMP routing functions in detail or which uses information not available to the IMP. The general properties of both strategies are usually similar, differing mainly in certain implementation details such as the availability of buffers or the constraint of counters and the need for the routing to quickly adapt to changes in IMP and circuit status.

The IMPs perform the routing computations using information received from other IMPs and local information such as the alive/dead state of its circuits. In the normal case of time varying loads, local information alone, such as the length of internal queues, is insufficient to provide an efficient routing strategy without assistance from the neighboring IMPs. It is possible to obtain sufficient information from the neighbors to provide efficient routing, with a small amount of computation needed per IMP and without each IMP requiring a topological map of the network. In certain applications where traffic patterns exhibit regularity, the use of a central controller might be preferable. However, for most applications which involve dynamically varying traffic flow, it appears that a central controller cannot be used more effectively than the IMPs to update routing tables if such a controller is constrained to derive its information via the net. It is also a less reliable approach to routing than to distribute the routing decisions among the IMPs.

The routing information cannot be propagated about the net in sufficient time to accurately characterize the instantaneous traffic flow. An efficient algorithm, therefore, should not focus on the movement of individual packets, but rather use topological or statistical information in the selection of routes. For example, by using an averaging procedure, the flow of traffic can be made to build up smoothly. This allows the routing algorithm ample time to adjust its tables in each IMP in advance of the build-up of traffic.

The scheme originally used in the ARPA network had each IMP select one output line per destination onto which to route packets. The line was chosen to be the one with minimum estimated time delay to the destination. The selection was updated every half second using minimum time estimates from the neighboring IMPs and internal estimates of the delay to each of the neighbors. Even though the routing algorithm only selects one line at a time per destination, two output lines will be used if a queue of packets waiting transmission on one line builds up before the routing update occurs and another line is chosen. Modifications to the scheme which allow several lines per destination to be used in an update interval (during which the routing is not changed) are possible using two or more time delay estimates to select the paths.

In practice, this approach has worked quite effectively with the moderate levels of traffic experienced in the net. For heavy traffic flow, this strategy may be inefficient, since the routing information is based on the length of queues, which we have seen can change much faster than the information about the change can be distributed. Fortunately, this information is still usable, although it can be substantially out of date and will not, in general, be helpful in making efficient routing decisions in the heavy traffic case.

A more intricate scheme, recently developed by BBN, allows multiple paths to be efficiently used even during heavy traffic.[18] Preliminary simulation studies indicate that it can be tailored to provide efficient routing in a network with a variety of heavy traffic conditions. This method separates the problem of defining routes onto which packets may be routed from the problem of selecting a route when a particular packet must be routed. By this technique, it is possible to send packets down a path with the fewest IMPs and excess capacity, or when that path is filled, the one with the next fewest IMPs and excess capacity, etc.

A similar approach to routing was independently derived by NAC using an idealized method that did not require the IMPs to participate in the routing decisions. Another approach using a flow deviation technique has recently been under study at UCLA.[13] The intricacies of the exact approach lead to a metering procedure that allows the overall network flow to be changed slowly for stability and to perturb existing flow patterns to obtain an increased flow. These approaches all possess, in common, essential ingredients of a desirable routing strategy.

Topological considerations

An efficient topological design provides a high throughput for a given cost. Although many measures of throughput are possible, a convenient one is the average amount of traffic that a single IMP can send into the network when all other IMPs are transmitting according to a specified traffic pattern. Often, it is assumed that all other IMPs are behaving identically and each IMP is sending equal amounts of traffic to each other IMP. The constraints on the topological design are the available common carrier circuits, the target cost or throughput, the desired reliability, and

TABLE I—23 Node 28 Link ARPA

Number of Circuits Failed	Number of Combinations to be Examined	Number of Cutsets
28	1	1
27	28	28
26	378	378
25	3276	3276
24	20475	20475
23	98280	98280
22	376740	376740
21	1184040	1184040
20	3108105	3108105
19	6906900	6906900
18	13123110	13123110
17	21474180	21474180
16	30421755	30421755
15	37442160	37442160
14	40116600	40116600
13	37442160	37442160
12	30421755	30421755
11	21474180	21474180
10	13123110	13123110
9	6906900	6906900
8	3108108	3108108
7	1184040	1184040
6	376740	349618
5	98280	≈70547
4	20475	≈9852
3	3276	827
2	378	30
1	28	0

the cost of computation required to perform the topological design.

Since, there was no clear specification of the amount of traffic that the network would have to accommodate initially, it was first constructed with enough excess capacity to accommodate any reasonable traffic requirements. Then as new IMPs were added to the system, the capacity was and is still being systematically reduced until the traffic level occupies a substantial fraction of the network's total capacity. At this point, the net's capacity will be increased to maintain the desired percentage of loading. At the initial stages of network design, the "two-connected" reliability constraint essentially determined a minimum value of maximum throughput. This constraint forces the average throughput to be in the range 10-15 kilobits per second per IMP, when 50 kilobit/sec circuits are used throughout the network, since two communication paths between every pair of IMPs are needed. Alternatively, if this level of throughput is required, then the reliability specification of "two-connectivity" can be obtained without additional cost.

Reliability computations

A simple and natural characterization of network reliability is the ability of the network to sustain communication between all operable pairs of IMPs. For design purposes, the requirement of two independent paths between nodes insures that at least two IMPs and/or circuits must fail before any pair of operable IMPs cannot communicate. This criterion is independent of the properties of the IMPs and circuits, does not take into account the "degree" of disruption that may occur and hence, does not reflect the actual availability of resources in the network. A more meaningful measure is the average fraction of IMP pairs that cannct communicate because of IMP and circuit failures. This calculation requires knowledge of the IMP and circuit failure rates, and could not be performed until enough operating data was gathered to make valid predictions.

To calculate network reliability, we must consider elementary network structures known as cutsets. A

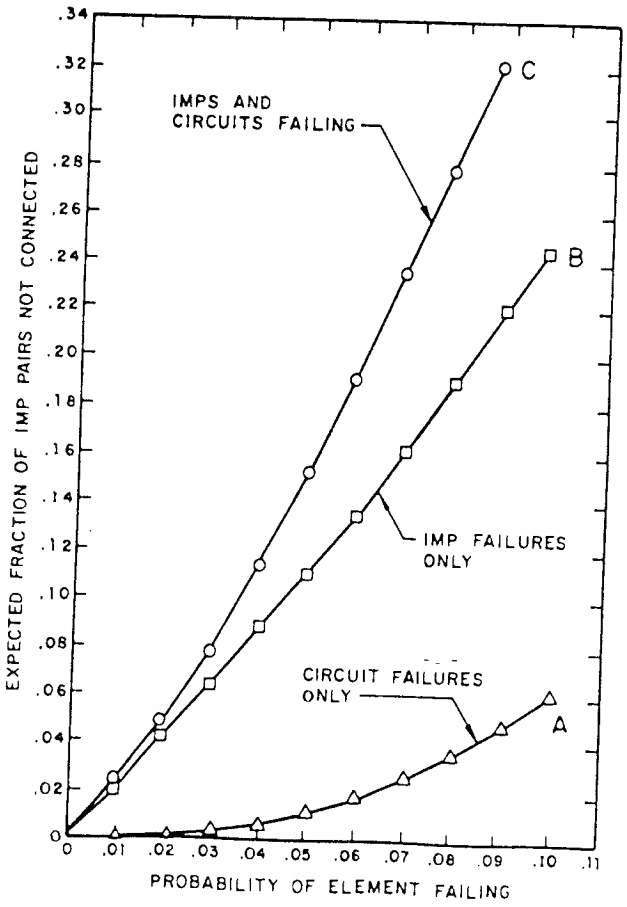


Figure 2—Network availability vs. IMP and circuit reliability

cutset is a set of circuits and/or IMPs whose removal from the network breaks all communication paths between at least two operable IMPs. To calculate reliability, it is often the case that all cutsets must be either enumerated or estimated. As an example, in a 23 IMP, 28 circuit ARPA Network design similar to the one shown in Figure 1(d), there are over twenty million ways of deleting only circuits so that the remaining network has at least one operable pair of IMPs with no intact communication paths. Table 1 indicates the numbers of cutsets in the 23 IMP network as a function of the number of circuits they contain.

A combination of analysis and simulation can be used to compute the average fraction of non-communicating IMP pairs. Detailed descriptions of the analysis methods are given in Reference 44 while their application to the analysis of the ARPANET is discussed in Reference 43. The results of an analysis of the 23 IMP version of the network are shown in Figure 2. The curve marked A shows the results under the assumption that IMPs do not fail, while the curve marked B shows the case where circuits do not fail. The curve marked C assumes that both IMPs and circuits fail with equal probability. In actual operation, the average failure probability of both IMPs and circuits is about 0.02. For this value, it can be seen that the effect of circuit failures is far less significant than the effect of IMP failures. If an IMP fails in a network with n IMPs, at least $n-1$ other IMPs cannot communicate with it. Thus, good network design cannot improve upon the effect directly due to IMP failures, which in the ARPANET is the major factor affecting the reliability of the communications. Further, more intricate reliability analyses which consider the loss of throughput capacity because of circuit failures have also been performed and these losses have been shown to be negligible.[28] Finally, unequal failure rates due to differences in line lengths have been shown to have only minor effects on the analysis and can usually be neglected.[27]

Topological optimization

During the computer optimization process, the reliability of the topology is assumed to be acceptable if the network is at least two-connected. The object of the optimization is to decrease the ratio of cost to throughput subject to an overall cost limitation. This technique employs a sophisticated network optimization program that utilizes circuit exchange heuristics, routing and flow analysis algorithms, to generate low cost designs. In addition, two time delay models were initially used to (1) calculate the throughput corresponding to an average time delay of 0.2 seconds, (2) estimate the packet rejection rate due to all buffers filling at an IMP. As experience with these models grew, the packet rejection rate was found to be negligible and the computation discontinued. The delay computation (Equation (7) in a later section) was subsequently first replaced by a heuristic calculation to speed the computation and later eliminated after it was found that time delays could be guaranteed to be acceptably low by preventing cutsets from being saturated. This "threshold" behavior is discussed further in the next section.

The basic method of optimization was described in Reference 12 while extensions to the design of large networks are discussed in Reference 9. The method operates by initially generating, either manually or by computer, a "starting network" that satisfies the overall network constraints but is not, in general, a low cost network. The computer then iteratively modifies the starting network in simple steps until a lower cost network is found that satisfies the constraints or the process is terminated. The process is repeated until no further improvements can be found. Using a different starting network can result in a different solution. However, by incorporating sensible heuristics and by using a variety of *carefully chosen* starting networks and some degree of man-machine interaction, "excellent" final networks usually result. Experience has shown that there are a wide variety of such networks with different topological structures but similar cost and performance.

The key to this design effort is the heuristic procedure by which the iterative network modifications are made. The method used in the ARPANET design involves the removal and addition of one or two circuits at a time. Many methods have been employed, at various times, to identify the appropriate circuits for potential addition or deletion. For example, to delete uneconomical circuits a straightforward procedure simply deletes single circuits in numerical order, reroutes traffic and reevaluates cost until a decrease in cost per megabit is found. At this point, the deletion is made permanent and the process begins again. A somewhat more sophisticated procedure deletes circuits in order of increasing utilization, while a more complex method attempts to evaluate the effect of the removal of any circuit before any deletion is attempted. The circuit with the greatest likelihood of an improvement is then considered for removal and so on.

There are a huge number of reasonable heuristics for circuit exchanges. After a great deal of experimentation with many of these, it appears that the choice of a particular heuristic is not critical. Instead, the speed and efficiency with which potential exchanges can be

investigated appears to be the limiting factor affecting the quality of the final design. Finally, as the size of the network increases, the greater the cost becomes to perform *any* circuit exchange optimization. Decomposition of the network design into regions becomes necessary and additional heuristics are needed to determine effective decompositions. It presently appears that these methods can be used to design relatively efficient networks with a few hundred IMPs while substantially new procedures will be necessary for networks of greater size.

The topological design requires a routing algorithm to evaluate the throughput capability of any given network. Its properties must reflect those of an implementable routing algorithm, for example, within the ARPANET. Although the routing problem can be formulated as a "multicommodity flow problem"[10] and solved by linear programming for networks with 20-30 IMPs,[8] faster techniques are needed when the routing algorithm is incorporated in a design procedure. The design procedure for the ARPA Network topology iteratively analyzes thousands of networks. To satisfy the requirements for speed, an algorithm which selects the least utilized path with the minimum number of IMPs was initially used.[12] This algorithm was later replaced by one which sends as much traffic as possible along such paths until one or more circuits approach a few percent of full utilization.[28] These highly utilized circuits are then no longer allowed to carry additional flow. Instead, new paths with excess capacity and possibly more intermediate nodes are found. The procedure continues until some cutset contains only nearly fully utilized circuits. At this point no additional flow can be sent. For design purposes, this algorithm is a highly satisfactory replacement for the more complicated multi-commodity approach. Using the algorithm, it has been shown that the throughput capabilities of the ARPA Network are substantially insensitive to the distribution of traffic and depend mainly only on the total traffic flow within the network.[8]

Analytic models of network performance

The effort to determine analytic models of system performance has proceeded in two phases: (1) the prediction of average time delay encountered by a message as it passes through the network, and (2) the use of these queueing models to calculate optimum channel capacity assignments for minimum possible delay. The model used as a standard for the average message delay was first described in Reference 21 where it served to predict delays in stochastic communication networks. In Reference 22, it was modified to describe the behavior of ARPA-like computer networks while in Reference 23 it was refined further to apply directly to the ARPANET.

The single server model

Queueing theory[20] provides an effective set of analytical tools for studying packet delay. Much of this theory considers systems in which messages place demands for transmission (service) upon a single communication channel (the single server). These systems are characterized by $A(\tau)$, the distribution of interarrival times between demands and $B(t)$, the distribution of service times. When the average demand for service is less than the capacity of the channel, the system is said to be stable.

When $A(\tau)$ is exponential (i.e., Poisson arrivals), and messages are transmitted on a first-come first-served basis, the average time T in the stable system is

$$T = \frac{\lambda \bar{t}^2}{2(1-\rho)} + \bar{t} \tag{1}$$

where λ is the average arrival rate of messages, $\bar{t}$ and $\bar{t}^2$ are the first and second moments of $B(t)$ respectively, and $\rho = \lambda \bar{t} < 1$. If the service time is also exponential,

$$T = \frac{\bar{t}}{1-\rho} \tag{2}$$

When $A(\tau)$ and $B(t)$ are arbitrary distributions, the situation becomes complex and only weak results are available. For example, no expression is available for T; however the following upper bound yields an excellent approximation[19] as $\rho \longrightarrow 1$:

$$T \leq \frac{\lambda(\sigma_a^2 + \sigma_b^2)}{2(1-\rho)} + \bar{t} \tag{3}$$

where σ_a^2 and σ_b^2 are the variance of the interarrival time and service time distributions, respectively.

Networks of queues

Multiple channels in a network environment give rise to queueing problems that are far more difficult to solve than single server systems. For example, the variability in the choice of source and destination for a message is a network phenomenon which contributes to delay. A principal analytical difficulty results from the fact that flows throughout the network are correlated. The basic approach to solving these stochastic network

problems is to *decompose* them into analyzable single-server problems which reflect the original network structure and traffic flow.

Early studies of queueing networks indicated that such a decomposition was possible;[20,21] however, those results do not carry over to message switched computer networks due to the correlation of traffic flows. In Reference 21 it was shown for a wide variety of communication nets that this correlation could be removed by considering the length of a given packet to be an independent random variable as it passes from node to node. Although this "independence" assumption is not physically realistic, it results in a mathematically tractable model which does not seem to affect the accuracy of the predicted time delays. As the size and connectivity of the network increases, the assumption becomes increasingly more realistic. With this assumption, a successful decomposition which permits a channel-by-channel analysis is possible, as follows.

The packet delay is defined as the time which a packet spends in the network from its entry until it reaches its destination. The average packet delay is denoted as T. Let Z_{jk} be the average delay for those packets whose origin is IMP j and whose destination is IMP k. We assume a Poisson arrival process for such packets with an average of γ_{jk} packets per second and an exponential distribution of packet lengths with an average of $1/\mu$ bits per packet. With these definitions, if γ is the sum of the quantities γ_{jk}, then[21]

$$T = \sum_{j,k} \frac{\gamma_{jk}}{\gamma} Z_{jk} \tag{4}$$

Let us now reformulate Equation (4) in terms of single channel delays. We first define the following quantities for the ith channel: C_i as its capacity (bits/second); λ_i as the average packet traffic it carries (packets/second); and T_i as the average time a packet spends waiting for and using the ith channel. By relating the $\{\lambda_i\}$ to the $\{\gamma_{jk}\}$ via the paths selected by the routing algorithm, it is easy to see that[21]

$$T = \sum_i \frac{\lambda_i}{\gamma} T_i \tag{5}$$

With the assumption of Poisson traffic and exponential service times, the quantities T_i are given by Equation (2). For an average packet length of $1/\mu$ bits, $\bar{t} = 1/\mu C_i$ seconds and thus

$$T_i = \frac{1}{\mu C_i - \lambda_i} \tag{6}$$

Thus we have successfully decomposed the analysis problem into a set of simple single-channel problems.

A refinement of the decomposition permits a non-exponential packet length distribution and uses Equation (1) rather than Equation (2) to calculate T_i; as an approximation, the Markovian character of the traffic is assumed to be preserved. Furthermore, for computer networks we include the effect of propagation time and overhead traffic to obtain the following equation for average packet delay[22,23]

$$T = K + \sum_i \frac{\lambda_i}{\gamma} \left[\frac{1}{\mu' C_i} + \frac{\lambda_i/\mu C_i}{\mu C_i - \lambda_i} + P_i + K \right] \tag{7}$$

Here, $1/\mu'$ represents the average length of a Host packet, and $1/\mu$ represents the average length of all packets (including acknowledgments, headers, requests for next messages, parity checks, etc.) within the network. The expression $1/\mu' C_i + [(\lambda_i/\mu C_i)/(\mu C_i - \lambda_i)] + P_i$ represents the average packet delay on the ith channel. The term $(\lambda_i/\mu C_i)/(\mu C_i - \lambda_i)$ is the average time a packet spends waiting at the IMP for the ith channel to become available. Since the packet must compete with acknowledgments and other overhead traffic, the overall average packet length $1/\mu$ appears in the expression. The term $1/\mu' C_i$ is the time required to transmit a packet of average length μ'. Finally: K is the nodal processing time, assumed constant, and for the ARPA IMP approximately equal to 0.35 ms; P_i is the propagation time on the ith channel (about 20 ms for a 3000 mile channel).

Assuming a relatively homogeneous set of C_i and P_i, no individual term in the expression for delay will dominate the summation until the flow λ_i/μ in one channel (say channel i_0) approaches the capacity C_{i_0}. At that point, the term T_{i_0}, and hence T will grow rapidly. The expression for delay is then dominated by one (or more) terms and exhibits a threshold behavior. Prior to this threshold, T remains relatively constant.

The accuracy of the time delay model, as well as this threshold phenomenon was demonstrated on a 19 node network[14] and on the ten node ARPA net derived from Figure 1(c) by deleting the rightmost five IMPs. Using the routing procedure described in the last section[28] and equal traffic between all node pairs, the channel flows λ_i were found for the ten node net and the delay curves shown in Figure 3 were obtained. Curve A was obtained with fixed 1000 bit packets,* while curve B was generated for exponentially distributed variable length packets with average size of 500 bits. In both cases A and B, all overhead factors were ignored. Note that the delay remains small until a

* In case A, the application of Equation (1) allows for constant packet lengths (i.e., zero variance).

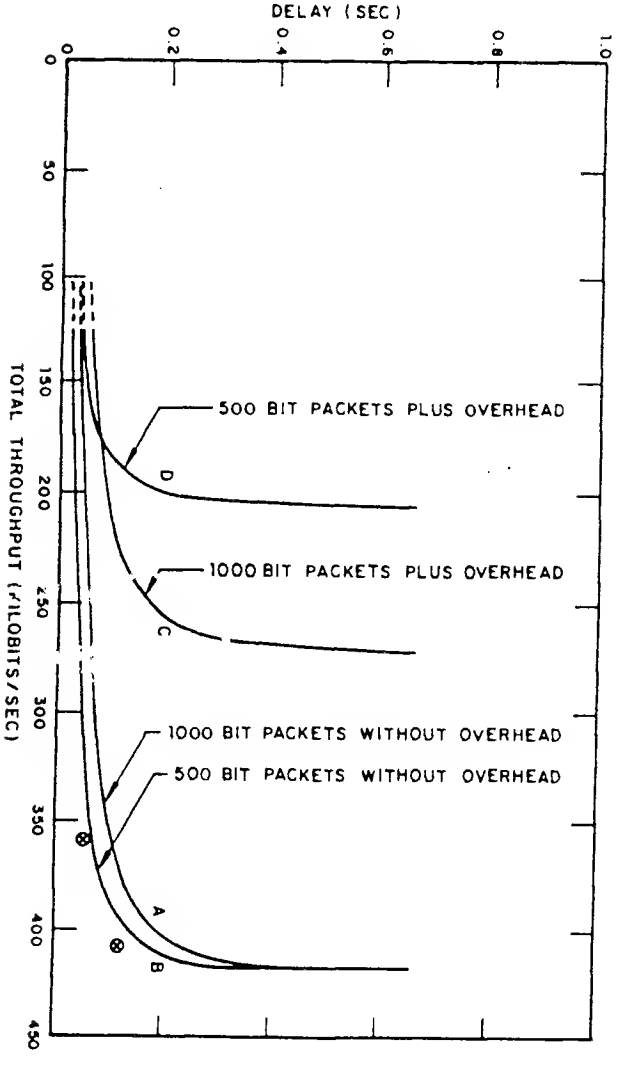


Figure 3—Delay vs. throughput

total throughput slightly greater than 400 kilobits/second is reached. The delay then increases rapidly. Curves C and D respectively represent the same situations when the overhead of 136 bits per packet and per RFNM and 152 bits per acknowledgment are included. Notice that the total throughput per IMP is reduced to 250 kilobits/second in case C and to approximately 200 kilobits/second in case D.

In the same figure, we have illustrated with x's the results of a simulation performed with a realistic routing and metering strategy. The simulation omitted all network overhead and assumed fixed lengths of 1000 bits for all packets.

It is difficult to develop a practical routing and flow control procedure that will allow each IMP to input identical amounts of traffic. To compare the delay curve A with the points obtained by simulation, the curve should actually be recomputed for the slightly skewed distribution that resulted. It is notable that the delay estimates from the simulation (which used a dynamic routing strategy) and the computation (which used a static routing strategy and the time delay formula) are in close agreement. In particular, they both accurately determined the vertical rise of the delay curve in the range just above 400 kilobits/second, the formula by predicting infinite delay and the simulation by rejecting the further input of traffic.

In practice and from the analytic and simulation studies of the ARPANET, the average queueing delay is observed to remain small (almost that of an unloaded net) and well within the design constraint of 0.2 seconds until the traffic within the network approaches the capacity of a cutset. The delay then increases rapidly. *Thus, as long as traffic is low enough and the routing adaptive enough to avoid the premature saturation of cutsets by guiding traffic along paths with excess capacity, queueing delays are not significant.*

Channel capacity optimization

One of the most difficult design problems is the optimal selection of capacities from a *finite* set of options. Although there are many heuristic approaches to this problem, analytic results are relatively scarce. (For the specialized case of centralized networks, an algorithm yielding optimal results is available.[11]) While it is possible to find an economical assignment of discrete capacities for, say, a 200 IMP network, very little is known about the relation between such capacity assignments, message delay, and cost.

To obtain theoretical properties of optimal capacity assignments, one may ignore the constraint that capacities are obtainable only in discrete sizes. In Reference 21 such a problem was posed where the network topology and average traffic flow were assumed to be known and fixed and an optimal match of capacities to traffic flow was found. Also, the traffic was assumed to be Markovian (Poisson arrivals and exponential packet lengths) and the independence assumption and decomposition method were applied. For each channel, the capacity C_i was found which minimized the average message delay T, at a fixed total system cost D. Since λ_i/μ is the average bit rate on the ith channel, the solution to *any* optimal assignment problem must provide more than this minimal capacity to each channel. This is clear since both Equations (6) and (7) indicate that T_i will become arbitrarily large with less than (or equal to) this amount of capacity. It is not critical exactly how the *excess* capacity is

assigned, as long as $C_i > \lambda_i/\mu$. Other important parameters and insights have been identified in studying the continuous capacity optimization problem. For example, the number of excess dollars, D_e, remaining after the minimum capacity λ_i/μ is assigned to each channel is of great importance. As $D_e \to 0$, the average delay must grow arbitrarily large. In this range, the critical parameters become ρ and $\bar{n}$ where $\rho = \gamma/\mu C$ is the ratio of the rate γ/μ at which bits enter the network to the rate C at which the net can handle bits and $\bar{n} = \lambda/\gamma$, where $\lambda = \sum \lambda_i$ is the total rate at which packets flow within the net. The quantity ρ represents a dimensionless form of network "load" whereas $\bar{n}$ is easily shown to represent the average path length for a packet.

As the load ρ approaches $1/\bar{n}$, the delay T grows very quickly, and this point $\rho = 1/\bar{n}$ represents the maximum load which the network can support. If capacities are assigned optimally, all channels saturate simultaneously at this point. In this formulation $\bar{n}$ is a design parameter which depends upon the topology and the routing procedure, while ρ is a parameter which depends upon the input rate and the total capacity of the network. In studying the ARPANET[23] a closer representation of the actual tariffs for high speed telephone data channels used in that network was provided by setting $D = \sum_i d_i C_i$ where $0 \leq \alpha \leq 1$.* This approach requires the solution of a non-linear equation by numerical techniques. On solving the equation, it can be shown that the packet delay T varies insignificantly with α for $.3 \leq \alpha \leq 1$. This indicates that the closed form solution discussed earlier with $\alpha = 1$ is a reasonable approximation to the more difficult non-linear problem. These continuous capacity studies have application to general network studies (e.g., satellite communications)[33] and are under continued investigation.[25,26,46]

In practice, the selection of channel capacities must be made from a small finite set. Although some theoretical work has been done in this case by approximating the discrete cost-capacity functions by continuous ones, much remains to be done.[13,25] Because of the discrete capacities and the time varying nature of network traffic, it is *not* generally possible to match channel capacities to the anticipated flows within the channels. If this were possible, all channels would saturate at the same externally applied load. Instead, capacities are assigned on the basis of reasonable estimates of average or peak traffic flows. It is the responsibility of the routing procedure to permit the traffic to adapt to the available capacity.[14] Often two IMP sites will engage in heavy communication and thus saturate one or more critical network cutsets. In such cases, the routing will not be able to send additional flow across these cuts. The network will therefore experience "premature" saturation in one or a small set of channels leading to the threshold behavior described earlier.

* Of course the tariffs reflect the discrete nature of available channels. The use of the exponent α provides a continuous fit to the discrete cost function. For the ARPANET, $\alpha \cong .8$.

DISCUSSION

A major conclusion from our experience in network design is that message switched networks of the ARPA type are no longer difficult to specify. They may be implemented straightforwardly from the specifications; they can be less expensive than other currently available technical approaches; they perform remarkably well as a communication system for interconnecting time-sharing and batch processing computers and can be adapted to directly handle teletypes, displays and many other kinds of terminal devices and data processing equipment.[16,30]

The principal tools available for the design of networks are analysis, simulation, heuristic procedures, and experimentation. Analysis, simulation and heuristics have been the mainstays of the work on modeling and topological optimization while simulation, heuristic procedures and experimental techniques have been the major tools for the actual network implementation. Experience has shown that all of these methods are useful while none are all powerful. The most valuable approach has been the simultaneous use of several of these tools.

Each approach has room for considerable improvement. The analysis efforts have not yet yielded results in many important areas such as routing. However, for prediction of delay, this approach leads to a simple threshold model which is both accurate and understandable. Heuristic procedures all suffer from the problem that it is presently unclear how to select appropriate heuristics. It has been the innovative use of computers and analysis that has made the approach work well. For designing networks with no more than a few hundred IMPs, present heuristics appear adequate but a good deal of additional work is required for networks of greater size. Simulation is a well developed tool that is both expensive to apply and limited in the overall understanding that it yields. For these reasons, simulation appears to be most useful only in validating models, and in assisting in detailed design decisions such as the number of buffers that an IMP should contain. As the size of networks continues to grow, it appears that simulation will become virtually useless as a total design tool. The ultimate standard by which all models and

conclusions can be tested is experimentation. Experimentation with the actual network is conceptually relatively straightforward and very useful. Although, experiments are often logistically difficult to perform, they can provide an easy means for testing models, heuristics and design parameters.

The outstanding design problems currently facing the network designer are to specify and determine the properties of the routing, flow control and topological structure for large networks. This specification must make full use of a wide variety of circuit options. Preliminary studies indicate that initially, the most fruitful approaches will be based on the partitioning of the network into regions, or equivalently, constructing a large network by connecting a number of regional networks. To send a message, a Host would specify both the destination region and the destination IMP in that region. No detailed implementation of a large network has yet been specified but early studies of their properties indicate that factors such as cost, throughput, delay and reliability are similar to those of the present ARPANET, if the ARPA technology is used.[9]

Techniques applicable to the design of large networks are presently under intensive study. These techniques appear to split into the same four categories as small network design but approaches may differ significantly. For example, large nets are likely to demand the placement of high bandwidth circuits at certain key locations in the topology to concentrate flow. These circuits will require the development of a high speed IMP to connect them into the net. It is likely that this high speed IMP will have the structure of a high speed multiplexor, and may require several cooperating processors to obtain the needed computer power for the job. Flow control strategies for large networks seem to extrapolate nicely from small network strategies if each region in the large network is viewed as a node in a smaller network. However, this area will require additional study as will the problem of specifying effective adaptive routing mechanisms. Recent efforts indicate that efficient practical schemes for small networks will soon be available. These schemes seem to be applicable for adaptive routing and flow control in networks constructed from regional subnetworks. The development of practical algorithms to handle routing and flow control is still an art rather than a science. Simulation is useful for studying the properties of a given heuristic, but intuition still plays a dominant role in the system design.

Several open questions in network design presently are: (1) what structure should a high bandwidth IMP have; (2) how can full use be made of a variety of high bandwidth circuits; (3) how should large networks be partitioned for both effective design and operation; and (4) what operational procedures should large networks follow? Much work has already been done in these areas but much more remains to be done. We expect substantial progress to be achieved in the next few years, and accordingly, the increased understanding of the properties of message switched networks of all sizes.

ACKNOWLEDGMENT

The ARPA Network is in large part the conception of Dr. L. G. Roberts of the Advanced Research Projects Agency to whom we owe a debt of gratitude for his support and encouragement. We also acknowledge the helpful contributions of S. Crocker and B. Dolan of ARPA. At BBN, NAC, and UCLA many individuals, too numerous to list, participated in the network effort and we gratefully acknowledge their contributions.

REFERENCES

1 P BARAN S BOEHM P SMITH
On distributed communications
Series of 11 reports by Rand Corporation Santa Monica California 1964

2 *"Specifications for the interconnection of a Host and an IMP*
BBN Report No 1822 1971 revision

3 S CARR S CROCKER V CERF
Host-Host communication protocol in the ARPA network
SJCC 1970 pp 589-597

4 S CROCKER et al
Function oriented protocols for the ARPA network
SJCC 1972 in this issue

5 D W DAVIES
The control of congestion in packet switching networks
Proc of the Second ACM IEEE Symposium on problems in the Optimization of Data Communications Systems Palo Alto California Oct 1971 pp 46-49

6 D FARBER K LARSON
The architecture of a distributed computer system—An informal description
University of California Irvine Information and Computer Science Technical Report #11 1970

7 W D FARMER E E NEWHALL
An experimental distribution switching system to handle bursty computer traffic
Proc of the ACM Symposium on Problems in the Optimization of Data Communication Systems 1969 pp 1-34

8 H FRANK W CHOU
Routing in computer networks
Networks John Wiley 1971 Vol 1 No 2 pp 99-112

9 H FRANK W CHOU
Cost and throughput in computer-communication networks
To appear in the Infotech Report on the State of the Art of Computer Networks 1972

10 H FRANK I T FRISCH
Communication transmission and transportation networks
Addison Wesley 1972

Function-oriented protocols for the ARPA Computer Network

by STEPHEN D. CROCKER
Advanced Research Projects Agency
Arlington, Virginia

JOHN F. HEAFNER
The RAND Corporation
Santa Monica, California

ROBERT M. METCALFE
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

JONATHAN B. POSTEL
University of California
Los Angeles, California

INTRODUCTION

Much has been said about the mechanics of the ARPA Computer Network (ARPANET) and especially about the organization of its communications subnet.[1,2,3,4,5] Until recently the main effort has gone into the implementation of an ARPANET user-level communications interface. Operating just above the communications subnet in ARPANET HOST Computers, this ARPANET interface is intended to serve as a foundation for the organization of function-oriented communications.[6,7] See Figures 1 and 2 for our view of a computer system and the scheme for user-level process-to-process communications. It is now appropriate to review the development of protocols which have been constructed to promote particular substantive uses of the ARPANET, namely function-oriented protocols.

We should begin this brief examination by stating what we mean by the word "protocol" and how protocols fit in the plan for useful work on the ARPANET. When we have two processes facing each other across some communication link, the protocol is the set of their agreements on the format and relative timing of messages to be exchanged. When we speak of a protocol, there is usually an important goal to be fulfilled. Although any set of agreements between cooperating (i.e., communicating) processes is a protocol, the protocols of interest are those which are constructed for general application by a large population of processes in solving a large class of problems.

In the understanding and generation of protocols there are two kinds of distinctions made. Protocols in the ARPANET are *layered* and we speak of high or low level protocols. High level protocols are those most closely matched to functions and low level protocols deal with communications mechanics. The lowest level software protocols in the ARPANET involve reliable message exchange between ARPANET Interface Message Processors (IMPs).[2,5] A high level protocol is one with primitives closely related to a substantive use. At the lowest levels the contents of messages are unspecified. At higher levels, more and more is stated about the meaning of message contents and timing. The layers of protocol are shown in Figure 3.

A second way of structuring sets of protocols and their design is bound up in the word *factoring*. At any level of protocol are sets of format and timing rules associated with particular groupings of agreements. In the IMPs we find certain protocols pertaining to error handling, while others to flow control, and still others to message routing. At the ARPANET's user-level communications interface there are, among others, separable protocols associated with establishing connections and logical data blocking. These protocols do not nest, but join as modules at the same level.

Before moving on to consider the higher level function-oriented protocols, let us first make a few statements about underlying protocols. There are three lower level software protocols which nest in support of the user-level communications interface for the ARPANET. The lowest of these is the IMP-IMP protocol which provides for reliable communication among IMPs. This protocol handles transmission-error detection and correction, flow control to avoid message congestion, and routing. At the next higher level is the IMP-HOST protocol which provides for the passage of messages between HOSTs and IMPs in such a way as to create virtual communication paths between HOSTs. With IMP-HOST protocol, a HOST has operating rules which permit it to send messages to specified HOSTs on the ARPANET and to be informed of the dispensation of those messages. In particular, the IMP-HOST protocol constrains HOSTs in their transmissions so that they can make good use of available

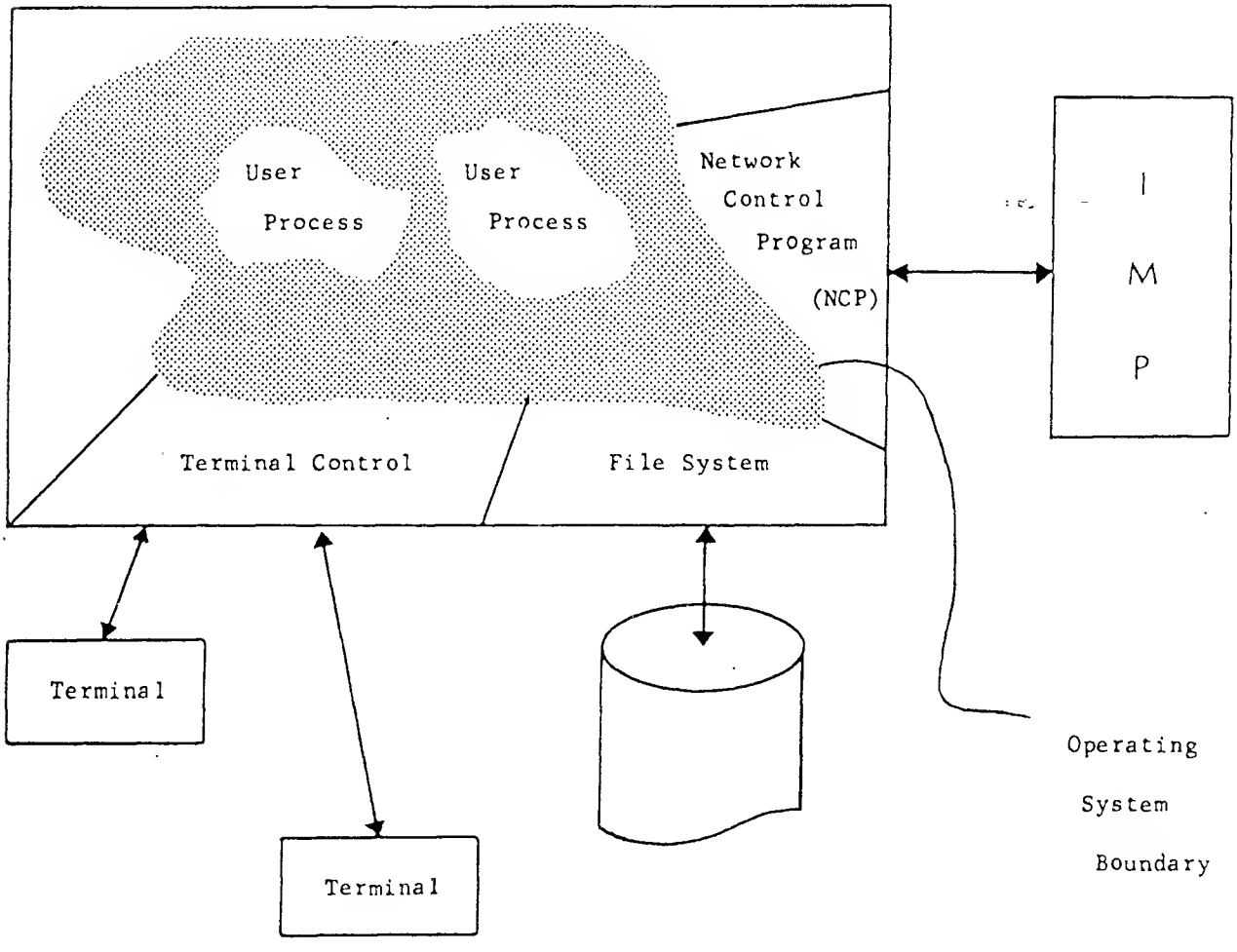


Figure 1—Our view of a computer system

communications capacity without denying such availability to other HOSTs.

The HOST-HOST protocol, finally, is the set of rules whereby HOSTs construct and maintain communication between processes (user jobs) running on remote computers. One process requiring communications with another on some remote computer system makes requests on its local supervisor to act on its behalf in establishing and maintaining those communications under HOST-HOST protocol.

In constructing these low levels of protocol it was the intention to provide user processes with a general set of useful communication primitives to isolate them from many of the details of operating systems and communications. At this user-level interface function-oriented protocols join as an open-ended collection of modules to make use of ARPANET capabilities.

The communications environment facing the designers of function-oriented protocols in the ARPANET is essentially that of a system of one-way byte-oriented connections. Technically speaking, a "connection" is a pair: a "send socket" at one end and a "receive socket" at the other. Primitives provided at the user-level interface include:

1. Initiate connection (local socket, foreign socket),
2. Wait for connection (local socket),
3. Send, Receive (local socket, data),
4. Close (local socket),
5. Send interrupt signal (local socket).

Processes in this virtual process network can create connections and transmit bytes. Connections are subject to HOST-HOST flow control and the vagaries of timing in a widely distributed computing environment, but care has been taken to give user processes control over their communications so as to make full use of network parallelism and redundancy. The kind of agreements which must be made in the creation of

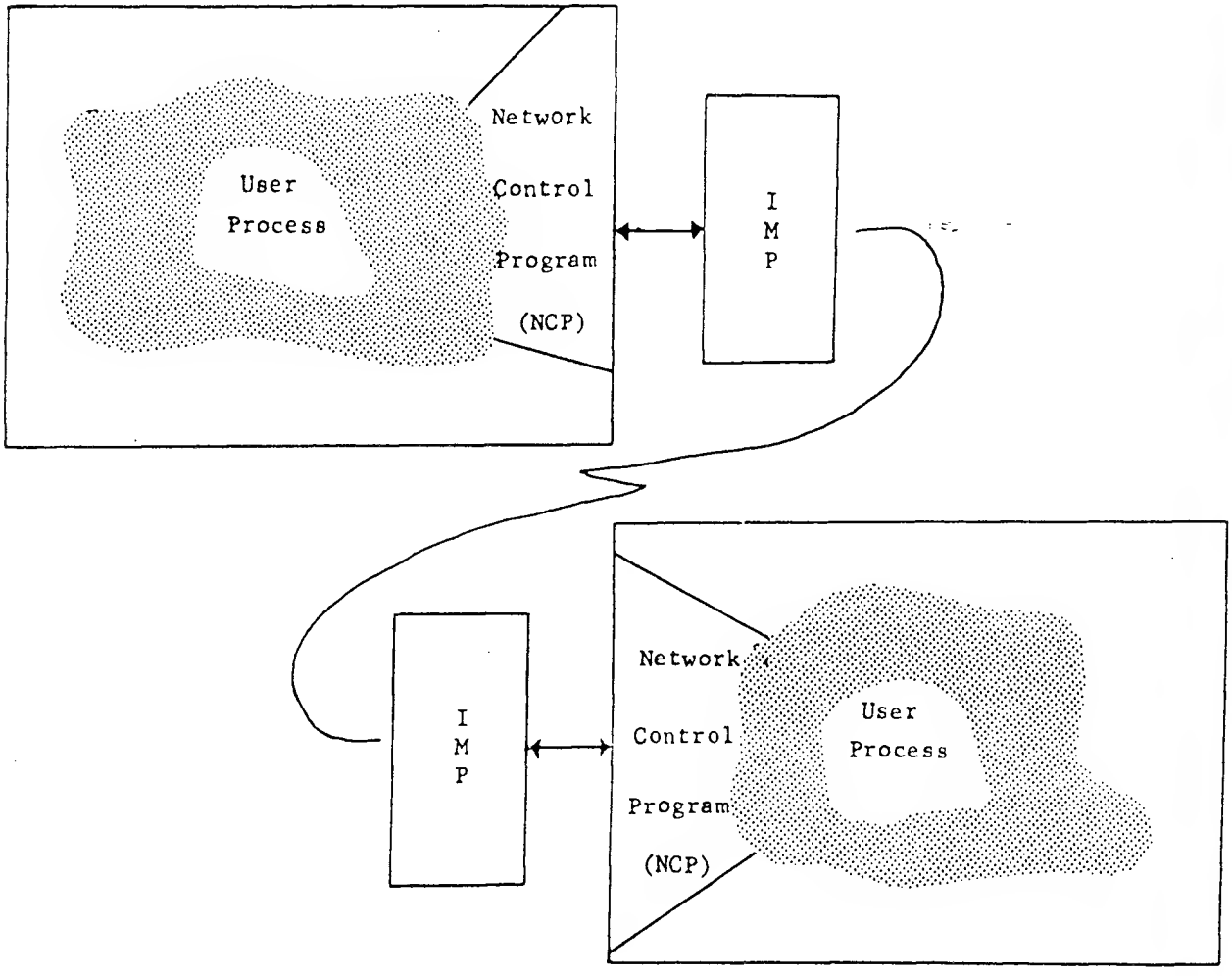


Figure 2—Two communicating processes

function-oriented protocols relate to rules for establishing connections, to the timing rules which govern transmission sequences, and to the content of the byte-streams themselves.

USE OF REMOTE INTERACTIVE SYSTEMS

The application which currently dominates ARPANET activity is the remote use of interactive systems. A Telecommunications Network (TELNET) protocol is followed by processes cooperating to support this application.[8] A user at a terminal, connected to his local HOST, controls a process in a remote HOST as if he were a local user of the remote HOST. His local HOST copies characters between his terminal and TELNET connections over the ARPANET. We refer to the HOST where the user sits as the *using HOST*, and to the remote HOST as the *serving HOST*. See Figure 4.

At the using HOST, the user must be able to perform the following functions through his TELNET user process ("user-TELNET"):

1. Initiate a pair of connections to a serving HOST,
2. Send characters to the serving HOST,
3. Receive characters from the serving HOST,
4. Send a HOST-HOST interrupt signal,
5. Terminate connections.

The user-TELNET needs to be able to distinguish between (1) commands to be acted on locally and (2) input intended for the serving HOST. An escape character is reserved to mark local commands. Conventions for the ARPANET Terminal IMP (TIP) user-TELNET are typical.[9]

In most using HOSTs, the above functions are provided by a user-TELNET which is a *user-level program*. A minimal user-TELNET need only implement the above functions, but several additional support func-